Aus dem Institut für Medizinische Biometrie und Informatik

KONZEPTION UND PROTOTYPISCHE IMPLEMENTIERUNG EINER INDIVIDUALISIERBAREN BENUTZEROBERFLÄCHE FÜR EINE BIOMEDIZINISCHE WEBAPPLIKATION

Diplomarbeit

zur Erlangung des Grades eines Dipl.-Inform. Med.
an der
Medizinischen Fakultät Heidelberg
der
Ruprecht-Karls-Universität

vorgelegt von

TINO NOACK

Referentin: Prof. Dr. Petra Knaup-Gregori
Korreferent: Prof. Dr. Martin Haag

03. Mai 2011

ZUSAMMENFASSUNG

In einem überregionalen Forschungsprojekt untersuchen Heidelberger und Hannoveraner Wissenschaftler Entstehungsmechanismen und neue Therapieansätze des Leberzellkarzinoms, einer der tödlichsten Tumorerkrankungen unserer Zeit. Die IT-Plattform pelican soll dem Forschungsverbund die softwaregestütze Analyse und den Austausch von Leberkrebs-Forschungsdaten ermöglichen. Bisher fehlte dazu ein geeignetes Oberflächenkonzept.

Deswegen wurde eine Benutzerschnittstelle entworfen, die in der Lage ist, heterogene Daten und Funktionen darzustellen und dabei flexibel an die Nutzerbedürfnisse angepasst werden kann. Es konnte anhand einer prototypischen Benutzungsoberfläche gezeigt werden, dass sich das Konzept praktisch umsetzen lässt. Mit Hilfe der in einem Evaluierungsprozess ausgewählten Portalsoftware GridSphere wurden mehrere Datendienste als Module in die pelican-Applikation integriert.

Als besondere Herausforderungen haben sich die Navigation innerhalb der Oberflächenmodule sowie der Informationsaustausch zwischen den Komponenten herausgestellt. Die erarbeiteten Lösungsansätze wurden dokumentiert und dienen als Basis für die Integration beliebiger neuer Dienste in die Plattform. Zukünftige Entwicklungen können der Krebsforschung weltweit in Form von standardkonformen Diensten im caGrid Umfeld zur Verfügung gestellt werden.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [Knuth 1974]

DANKSAGUNG

Ich danke Prof. Petra Knaup-Gregori für die exzellente Betreuung meiner Diplomarbeit und die zahlreichen Tips, um im Dschungel des wissenschaftlichen Arbeitens nicht verloren zu gehen.

Matthias Ganzinger ist stets ein wertvoller Ansprechpartner gewesen und treibt das pelican Projekt mit seinen Ideen und seinem großen Erfahrungsschatz unermüdlich voran.

Bei Dr. Thomas Longerich, Dr. Sven Diederichs und Olaf Neumann bedanke ich mich herzlich für die zur Verfügung gestellten Datensätze und ihre Auskünfte zu medizinischen Themen.

Mein besonderer Dank gilt den Menschen, die mich durch die Zeit der Diplomarbeit begleitet haben und auf die eine oder andere Weise zu ihrem erfolgreichem Abschluss beigetragen haben.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

API	Application Programming Interface
ACGH	Array Comparative Genomic Hybridization
caBIG	cancer Biomedical Informatics Grid
CMS	Content Management System
CSS	Cascading Style Sheet
DBMS	Datenbankmanagementsystem
DFG	Deutsche Forschungsgemeinschaft
DKFZ	Deutsches Krebsforschungszentrum Heidelberg
HCC	Hepatocellular Carcinoma
HZI	Helmholtz Zentrum für Infektionsforschung Braunschweig
IDE	integrated development environment
JSF	JavaServer Faces
JSP	JavaServer Pages
LDAP	Lightweight Directory Access Protocol
MHH	Medizinische Hochschule Hannover
NCI	National Cancer Institute
OASIS	Organization for the Advancement of Structured Information Standards
OSS	Open Source Software
pelican	platform enhancing livercancer networked research
RNA	Ribonukleinsäure
SFB	Sonderforschungsbereich
SOA	Serviceorientierte Architektur
SQL	Structured Query Language
TRR	Transregio
UKHD	Uniklinikum Heidelberg
W3C	World Wide Web Consortium

1 EINLEITUNG

GEGENSTAND

Seit Januar 2010 fördert die Deutsche Forschungsgemeinschaft (DFG) einen transregionalen Sonderforschungsbereich (SFB) mit dem Titel „Leberkrebs - Von der molekularen Pathogenese zur zielgerichteten Therapie“. Partner sind das Uniklinikum Heidelberg (UKHD), die Medizinische Hochschule Hannover (MHH), das Deutsche Krebsforschungszentrum Heidelberg (DKFZ) und das Helmholtz Zentrum für Infektionsforschung Braunschweig (HZI). Ziel des regionsübergreifenden Forschungsbereiches soll es sein, die zugrundeliegenden Mechanismen genauer zu untersuchen, die zu einem Leberzellkarzinom führen können. Möglicherweise lassen sich aus diesem Wissen neue erfolgversprechende Therapieansätze ableiten.

Die Arbeit der Forscher soll in Zukunft durch eine integrierte IT-Plattform unterstützt werden, die einen projektübergreifenden Austausch von Daten, Ergebnissen und Dokumenten ermöglicht. Darüber hinaus sollen informationstechnische Werkzeuge zur Auswertung der vorliegenden Daten zur Verfügung gestellt werden. Die geplante Verwendung einer Serviceorientierten Architektur (SOA) soll eine flexible Anpassung der Software an die Bedürfnisse der Benutzer gewährleisten. Angestrebt wird die Nutzung von IT-Werkzeugen und Ideen aus dem cancer Biomedical Informatics Grid (caBIG), welches die Sammlung, Verbreitung und Analyse von Daten und Wissen zum Thema Krebs unterstützt.

PROBLEMATIK UND MOTIVATION

Die Möglichkeiten moderner IT-Lösungen bleiben derzeit noch weitgehend ungenutzt im Arbeitsablauf der biomedizinischen Forschung. Daten und Ergebnisse werden oft zeitintensiv per Hand erfasst und ausgewertet. Eine Integration der Daten für eine gemeinsame, projektweite und projektübergreifende Nutzung findet häufig nicht statt. Gründe hierfür sind die Vielfalt wissenschaftlicher Fragestellungen innerhalb eines Forschungsverbundes wie des Transregio (TRR), die Heterogenität der verwendeten Datenstrukturen und -quellen sowie zu geringe Mittel für IT-Personal.

Um die Arbeit der Forscher sinnvoll unterstützen zu können, muss die IT-Plattform den speziellen Anforderungen der unterschiedlichen Nut-

zer durch eine individualisierbare Benutzeroberfläche gerecht werden. Dazu bedarf es eines geeigneten Oberflächenkonzepts, das einerseits komplexe Funktionen und Dienste flexibel integrieren kann, aber dennoch intuitive Bedienbarkeit und individuelle Konfiguration auch für nicht-technische Benutzer ermöglicht.

Ein konkretes Beispiel: Innerhalb des Projekts wurde bereits ein Prototyp entwickelt, welcher Daten über genetisch veränderte Sequenzen in Chromosomen tabellarisch darstellt. Dieses Datenmodul soll nun im Kontext anderer Dienste der SOA-Umgebung in die Gesamtoberfläche eines Nutzers integriert werden, der eine entsprechende Funktionalität für die Auswertung seiner Daten benötigt.

Die besondere Herausforderung der geplanten IT-Plattform ist jedoch, dass die zu entwickelnde Oberfläche die Darstellung von zukünftigen Modulen und Diensten unabhängig von konkreten Datenquellen und Datentypen erlauben muss. Es existieren bereits IT-Lösungen in Form von Portalsoftware, welche einzelne Benutzerschnittstellenkomponenten oder Werkzeuge für die Integration von SOA-Diensten in die Applikationsoberfläche zur Verfügung stellen. Deren spezifische Eignung muss jedoch erst sorgfältig geprüft werden.

PROBLEMSTELLUNG

Für die geplante Integrationsplattform wurde noch keine Benutzeroberfläche entworfen, die in der Lage ist, durch die sinnvolle Nutzung moderner Webtechnologien flexibel auf die Bedürfnisse unterschiedlicher Benutzer zu reagieren. Bislang wurde im Rahmen des Projekts keine geeignete Softwarelösung zur Umsetzung eines entsprechenden Oberflächenkonzepts ermittelt. Die Eignung des Portalansatzes für die pelican-Anwendung, in die sich später beliebige Einzelmodule (als Darstellung von Diensten und Funktionen) einbinden lassen sollen, wurde bisher nicht in der Praxis gezeigt.

ZIELSETZUNG

- Ziel 1: Evaluation und Auswahl einer Portalsoftware für die Verwendung innerhalb der pelican-Applikation.
- Ziel 2: Entwicklung eines flexiblen Benutzerschnittstellenkonzepts für die Integrationsplattform.
- Ziel 3: Validierung des Konzepts durch Einbinden bestehender Funktionen in eine modulare Anwendungsoberfläche.

FRAGE- UND AUFGABENSTELLUNG

Fragen und Aufgaben zu Ziel 1:
1.1 Welche Methodik eignet sich für eine systematische Evaluation von Portalsoftware?
1.2 Welchen funktionellen und technischen Anforderungen unterliegt die Benutzerschnittstelle?
1.3 Welche Portalsoftware-Lösungen sollen in die Evaluation einbezogen werden?
1.4 Erstellung einer Bewertungsmatrix unter Berücksichtigung der Kriterien aus 1.2
1.5 Welche Softwarelösung ist für den Einsatz in der Integrationsplattform am ehesten geeignet?

Fragen und Aufgaben zu Ziel 2:
2.1 Erstellung eines flexiblen Benutzerschnittstellenkonzepts.
2.2 Lässt sich das Konzept unter Verwendung der ausgewählten Portalsoftware prototypisch umsetzen?

Fragen und Aufgaben zu Ziel 3:
3.1 Lassen sich bereits implementierte pelican-Funktionen als Module in die Oberfläche integrieren?
3.2 Welche applikationsspezifischen Besonderheiten traten auf?

2 GRUNDLAGEN

2.1 Über den Stand der Biowissenschaften

> „Bioinformatics is a discipline based on a wealth of diverse, complex and distributed data resources [Goble und Stevens 2008]."

Das einführende Zitat beschreibt treffend die Herausforderungen, die die moderne medizinische Forschung an die Medizin- und Bioinformatik heute stellt. Die enorme Rechenleistung heutiger Computer und fortgeschrittene Technologien wie Microarrays ermöglichen eine Sammlung von Daten in beispielloser Dimension. Die gigantische Menge an unstrukturierten, verschiedenartigen und verteilten Daten trägt jedoch nicht automatisch zum besseren Verständnis der Materie bei. Es bedarf einer sorgfältigen Ordnung, Deskription und Verknüpfung, um die Forschung der Biowissenschaftler sinnvoll unterstützen zu können.

Die Anforderungen an ihre IT-Kompetenz sind in gleichem Maße wie das Volumen der generierten Daten gewachsen. Reichte es den Wissenschaftlern früher, einen Webbrowser bedienen zu können und mehr oder weniger regelmäßig per E-Mail zu kommunizieren, benötigen sie zur Beantwortung ihrer Forschungsfragen heute fast immer Spezialprogramme und Datenbanken. Nicht selten verlangt die Bedienung der Software Kenntnisse aus dem IT-Bereich, zum Beispiel zur Formulierung einer Datenbankabfrage per Structured Query Language (SQL). Darüber hinaus existieren viele spezialisierte Datenbanken mit jeweils eigenen Schnittstellen und Nomenklaturen und einer großen Menge an Redundanz untereinander. Dies erschwert die Formulierung übergeordneter Forschungsfragen, die als zukunftsweisend für die Weiterentwicklung der Biowissenschaften gelten.

Stellvertretend für alle Fachgebiete konstatieren Martone et al. [2004] deshalb: „Both neuroscientists and computer scientists are increasingly turning to more distributed architectures, where independent, distributed data resources can participate in larger, collaborative virtual data federations."

Die Dimension heutiger medizinischer Forschungsdaten verlangt demzufolge nach neuen Strukturen. Im Fokus steht nicht mehr die Leistung des einzelnen Wissenschaftlers oder Labors, sondern deren Beitrag zu größeren, interdisziplinären Wissensgemeinschaften. Stein [2008] hält derartige „Datenverbünde" gar für den Schlüssel zum Fortschritt:

„[...] interdisciplinary collaborations among experimental biologists, theorists, statisticians and computer scientists have become the key to making effective use of these data sets."

Die Ziele einer solchen „scientific cyberinfrastructure" seien, den Wissenschaftlern sowohl Informationen als auch die zum Verständnis der Daten notwendigen Werkzeuge zur Verfügung zu stellen und die Veröffentlichung und den Austausch der Ergebnisse und des erworbenen Wissens zu unterstützen. Gleichzeitig ist jedoch festzustellen: „[...] too many biologists have trouble accessing and using these electronic data sets and tools effectively [Stein 2008]."

Um die Ursachen hierfür verstehen zu können, sollte man sich zunächst vergegenwärtigen, was eine funktionierende IT-Infrastruktur ausmacht. Stein identifiziert folgende vier Kernkomponenten:

- die Datenbasis (Datenbanken wie PubMed oder ENSEMBL)
- rechenbetonte Komponenten (Hard- und Software wie das statistische Analysetool SPSS oder der BLAST Algorithmus zur Sequenzalignierung)
- Kommunikationskomponenten (Netzwerkverbindung, semantische und syntaktische Interoperabilität)
- menschliche Beteiligte (Aufbau, Nutzung und Pflege der Infrastruktur)

Um die biomedizinische Forschung effizient zu unterstützen, ist für jede dieser Komponenten ein systematisches IT-Management notwendig. Die Planungsphase umfasst eine Anforderungsanalyse, die Definition der Funktionalität und die Entscheidung für eine IT-Architektur. Nach der Implementierung muss das System gepflegt und nachhaltig bereitgestellt werden. Eine Schulung erhöht die Akzeptanz der zukünftigen Nutzer gegenüber der Anwendung und unterstützt bei der Integration des Werkzeugs in den bisherigen Arbeitsablauf.

2.2 LEBERKREBSFORSCHUNG

Das Leberzellkarzinom, oder englisch Hepatocellular Carcinoma (HCC), stellt mit weltweit über 620.000 Neuerkrankungen und knapp 600.000 verursachten Todesfällen im Jahr 2002 eine der tödlichsten Krebserkrankungen unserer Zeit dar [Parkin et al. 2005, S. 78]. Für die Vereinigten Staaten von Amerika wurde von 1995 bis 2000 eine 5-Jahres-Überlebensrate von lediglich 8,4% ermittelt [Hertl 2005]. Die Prognose ist wegen oft später Entdeckung und einer Vielzahl an möglichen Komplikationen äußerst schlecht.

Aus diesem Grund haben sich Mediziner und Forscher aus Heidelberg, Hannover und Braunschweig im Sonderforschungsbereich/Transregio 77 zusammengeschlossen. Durch gemeinsame, projektübergreifende Forschung zu verschiedenen Aspekten der Erkrankung Leberkrebs sollen neue Erkenntnisse gewonnen und somit bessere Therapien entwickelt werden. Sonderforschungsbereiche stellen dafür einen geeigneten Rahmen dar.

2.2.1 *Sonderforschungsbereiche*

Sonderforschungsbereiche sind auf die Dauer von bis zu zwölf Jahren angelegte Forschungseinrichtungen der Hochschulen, in denen Wissenschaftler und Wissenschaftlerinnen über die Grenzen ihrer jeweiligen Fächer, Institute, Fachbereiche und Fakultäten hinweg im Rahmen eines übergreifenden und wissenschaftlich exzellenten Forschungsprogramms zusammenarbeiten. Sie ermöglichen die Bearbeitung innovativer, anspruchsvoller, aufwändiger und langfristig konzipierter Forschungsvorhaben durch Konzentration und Koordination der in einer Hochschule vorhandenen Kräfte.

Programmvariante Transregio

Der Sonderforschungsbereich/Transregio wird von mehreren (in der Regel bis zu drei) Hochschulen gemeinsam beantragt. Die Beiträge der Verbundpartner sind für das gemeinsame Forschungsziel essentiell, komplementär und synergetisch. Die Förderung ermöglicht eine enge überregionale Kooperation zwischen Hochschulen und den dort Forschenden sowie eine Vernetzung und gemeinsame Nutzung der Ressourcen[1].

1 `http://www.dfg.de/foerderung/programme/koordinierte_programme/sfb/index.html`, zuletzt abgerufen am 20.12.2010

Sonderforschungsbereich/Transregio 77

Der SFB/TRR 77 besteht aus 24 Teilprojekten und gliedert sich in 4 Teilbereiche:

Der Forschungsbereich A („From Chronic Liver Disease to Tumor Formation“) erforscht die Mechanismen chronischer Lebererkrankungen, die Krebs hervorrufen oder begünstigen können. Ziel ist es, neue Präventionsstrategien zu entwickeln.

Bereich B („Basic Molecular Mechanisms and Novel Tactics“) konzentriert sich auf die Identifikation und Klassifizierung molekularer Vorgänge, die eine Krebserkrankung charakterisieren. So gefundene Marker und Mechanismen können Ziele neuer Therapieansätze werden.

Die Projekte des Bereiches C („Novel Approaches to Therapy“) untersuchen bereits identifizierte tumorspezifische Mechanismen, um bestehende Therapieverfahren sowie die gezielte Modifikation von Tumorzellen, so genanntes Targeting, zu optimieren.

Der Bereich Z umfasst die beiden Querschnittsprojekte des SFB/TRR 77. Ein zentrales administratives Teilprojekt (Z1) koordiniert sämtliche Aktivitäten des TRR und ist zuständig für alle finanziellen sowie externen Belange wie internationale Kooperationen und Pressemeldungen. Das andere Teilprojekt Z2 („Integrated Information Platform and Cross-Project Biostatistical Analyses“) stellt die informationstechnische Infrastruktur zur Verfügung und bietet zusätzliche Expertise bei statistischen Fragestellungen an. Die vorliegende Diplomarbeit entstand im Rahmen dieses Teilprojekts.

In den wissenschaftlichen Projekten werden große Mengen an Daten produziert, die sich in ihrer Struktur grundlegend unterscheiden können und sich deshalb nicht ohne Weiteres verknüpfen lassen. Im Teilprojekt B5 von Dr. Thomas Longerich werden beispielsweise Ergebnisse einer so genannten vergleichenden Genomhybridisierung (engl. Array Comparative Genomic Hybridization (ACGH)) analysiert. Diese in einem Microarrayexperiment erhobenen Daten ermöglichen den Nachweis von Veränderungen des menschlichen Erbguts mit sehr hoher Auflösung. Es lässt sich also feststellen, ob das untersuchte Erbgut im Vergleich zu einem anderen an bestimmten Stellen eine Mutation erfahren hat, zum Beispiel Krebsgewebe gegenüber gesundem Gewebe. Die Wissenshaftler suchen nun nach einer Möglichkeit, diese Informationen mit Erkenntnissen aus anderen Projekten in Verbindung zu bringen (welche womöglich mit Bilddaten anstatt Microarraydaten arbeiten).

2.2.2 *Das pelican-Projekt*

Dem Wunsch der Forscher nach einer gemeinsamen Arbeitsgrundlage soll in Form der integrierten Informationsplattform „pelican - **p**latform **e**nhancing **li**ver**ca**ncer **n**etworked research“ entsprochen werden, die derzeit im Z2 Projekt entwickelt wird. Sie ermöglicht einen Überblick über alle im Verbund verfügbaren Forschungsdaten und Ressourcen sowie eine projektübergreifende Auswertung von Forschungsergebnissen.

Die vielen unterschiedlichen Bedürfnisse und für die Forschungsfragen verwendete Datenformate zeigen jedoch die Komplexität des Projektes auf. Deswegen muss die Architektur der Informationsplattform besonders sorgfältig konzipiert werden. Einen vielversprechenden Ansatz stellt eine Serviceorientiert Architektur dar, deren Grundlagen im folgenden Kapitel näher erläutert werden.

2.3 SERVICEORIENTIERTE ARCHITEKTUR

Mitte der Neunziger Jahre etablierte sich mit der SOA ein Ansatz, der bewährte Konzepte vorheriger IT-Architekturen übernahm, bündelte und konsequent weiterentwickelte, um immer komplexere Softwareszenarien beherrschen zu können [Melzer 2008]. Unter einer Architektur versteht man im Bereich der Softwareentwicklung die Struktur eines Softwaresystems und die Beziehungen der einzelnen Komponenten untereinander [Balzert 2001].

Die namensgebenden, zentralen Komponenten einer SOA sind so genannte Dienste (engl. services). Die Grundidee ist, durch die Verwendung und Verknüpfung standardisierter Funktionsbausteine, den Diensten, eine „hochgradig normalisierte und schlanke Architektur [Erl 2010]“ zu entwickeln. Während bei traditionellen Softwareansätzen die Integration neuer Anwendungen eine große Herausforderung darstellt und die IT-Infrastruktur immer komplexer werden lässt, ist SOA stark auf Interoperabilität und Vermeidung von Redundanz ausgerichtet.

2.3.1 *Definition und Merkmale*

Derzeit gibt es keine allgemein gültige Definition des Begriffs SOA. Die Interpretation variiert nach Anforderung der jeweiligen Organisation, ist darüber hinaus oft entweder zu allgemein oder zu speziell. Die Organization for the Advancement of Structured Information Standards (OASIS) lieferte in ihrem SOA Referenzmodell von 2006 jedoch folgende, oft zitierte Definition:

> „Serviceorientierte Architektur ist ein Paradigma zur Organisation und Verwendung verteilter Funktionalität, deren Kontrolle unterschiedlichen Besitzern obliegen kann [OASIS 2006].“

Etwas konkreter wird Melzer [2008], der unter einer SOA eine Systemarchitektur versteht, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.

Erl [2010] beschreibt in „SOA - Entwurfsprinzipien für serviceorientierte Architektur“ acht grundlegende Merkmale des Konzepts.

1. **Standardisierter Servicevertrag.** Die öffentliche Schnittstelle eines Services (sprich, seine Beschreibung) muss bestimmte standardisierte Informationen über Funktion und Fähigkeiten des

Services bereitstellen. Art und Umfang dieser Informationen sind im Servicevertrag unter einzelnen Verbundpartnern festgelegt.

2. **Lose Kopplung.** Die Abhängigkeit der Dienste untereinander soll minimal sein. Der angestrebte Zustand ist, dass Dienste nur von der Existenz der anderen Dienste wissen und sonst keinerlei Abhängigkeiten bestehen.

3. **Abstraktion von Services.** Dieses Prinzip fordert, dass Dienste jegliche Informationen verbergen sollen, die über die Dienstbeschreibung beziehungsweise öffentliche Schnittstelle hinausgehen. Dies betrifft insbesondere die innere Programmlogik.

4. **Wiederverwendbarkeit von Services.** Die Logik wird in Dienste aufgeteilt, um wiederkehrende Anfragen ressourcenschonend zu verarbeiten. Das Ziel hierbei ist eine Einsparung von Entwicklungszeit und -kosten.

5. **Autonomie von Services.** Damit die Dienste zuverlässig funktionieren können, müssen sie jederzeit die Kontrolle über ihre interne Logik und Ressourcen behalten.

6. **Zustandslosigkeit von Services.** Je seltener die Dienste sich in einem konkreten Zustand befinden, desto ressourcenschonender ist ihre Verwaltung. Müsste sich ein Service für jede Anfrage merken, in welchem von beliebig vielen Zuständen er sich gerade befindet, führte das zwangsläufig zu Problemen bei der Verfügbarkeit unter starker Last (sprich, vielen Anfragen).

7. **Auffindbarkeit von Services.** Dienste sollen für die Nutzer leicht auffindbar und verständlich sein. Dafür müssen sie mit Metadaten ausgestattet sein, die eine effektive Suche ermöglichen (zum Beispiel in einem klassifizierten Dienstverzeichnis). Wünschenswert wäre neben einer Suche nach genauen Schlagworten (syntaktische Suche) vielmehr eine semantische Suche, welche die natürliche Sprache als Anfrage akzeptiert und den Sinn automatisch erfasst. Die Technologie dafür steckt allerdings noch in den Kinderschuhen.

8. **Kompositionsfähigkeit von Services.** Dienste sollen sich wirkungsvoll kombinieren lassen, um komplexe Szenarien darstellen zu können. Man nennt diese Verknüpfung auch Dienstkomposition. Sie ist am Beispiel eines fiktiven Arbeitsablaufs in Abbildung 2.1 dargestellt.

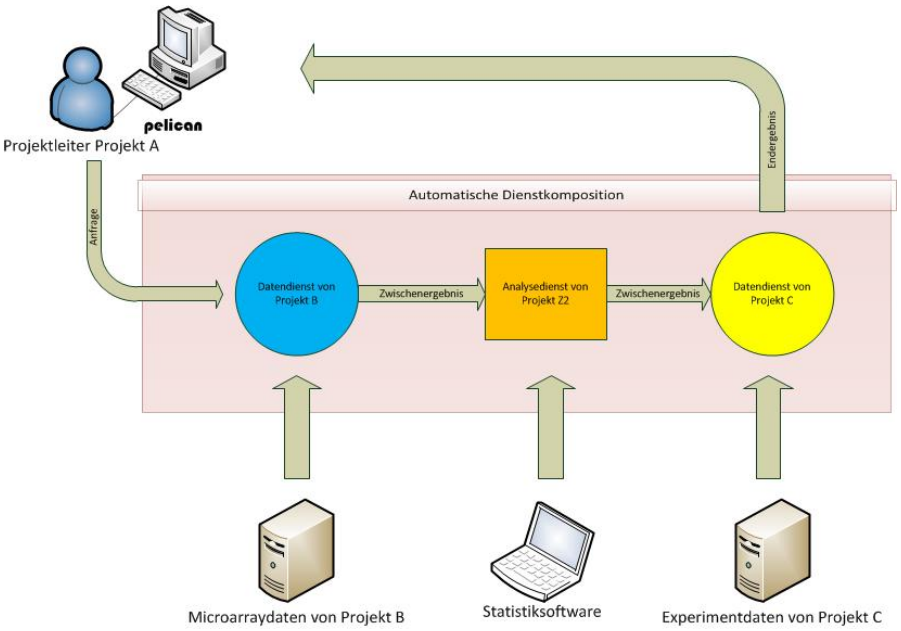

Abbildung 2.1: Beispielhafte Komposition von pelican Diensten. Projektleiter A kombiniert drei verschiedene Dienste der Anwendung für eine projektübergreifende Datenanalyse inklusive statistischer Auswertung.

2.3.2 *Rollen und Aktionen*

In einer SOA-Umgebung lassen sich nach Dostal [2005] drei verschiedene Rollen ausmachen.

Der *Dienstanbieter* stellt seinen (fertig implementierten) Service zur Verfügung und registriert ihn im gemeinsamen Dienstverzeichnis, vergleichbar mit einem Eintrag in die Gelben Seiten. Dieser Vorgang wird als Publishing beziehungsweise Veröffentlichen bezeichnet.

Im *Dienstverzeichnis*, auch Registry oder Repository genannt, sind die Schnittstellenbeschreibungen aller eingetragenen Dienste hinterlegt. Hier kann nach bestimmten Eigenschaften, Schlagworten oder Kategorien gesucht werden.

Der *Dienstnutzer* erfragt diese Eigenschaften im Dienstverzeichnis und sucht sich entsprechend seiner Fragestellung einen passenden Dienst heraus. Er erhält einen Verweis auf den Dienst und kann sich jetzt direkt an dessen Anbieter wenden, um den Service tatsächlich zu nutzen.

Das Zusammenwirken aller drei Rollen soll in Abbildung 2.2 verdeutlicht werden.

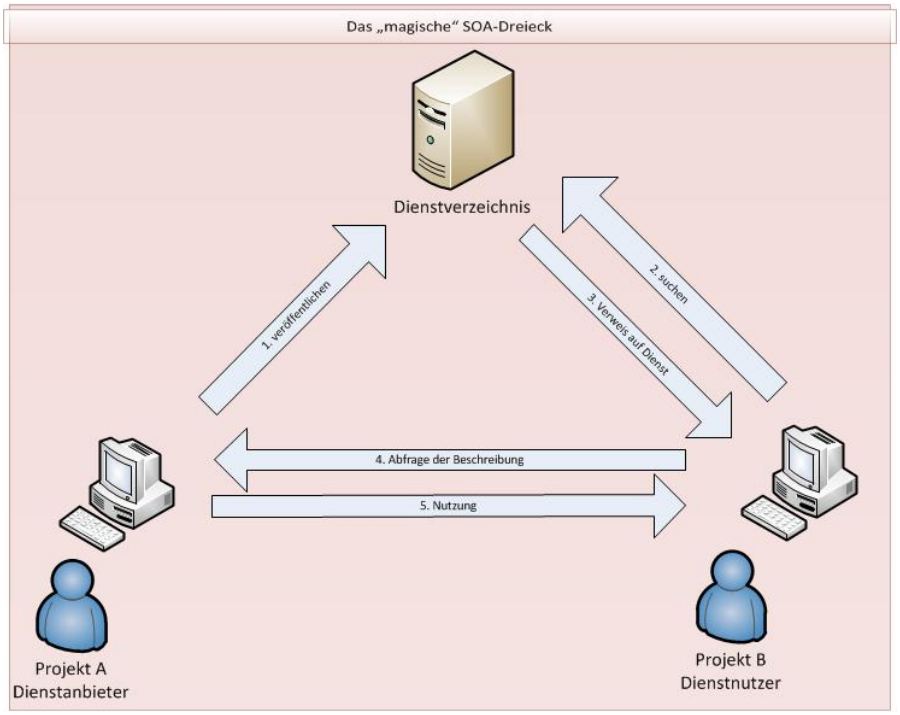

Abbildung 2.2: Rollen innerhalb einer SOA, beispielhaft dargestellt am Zusammenspiel zweier fiktiver pelican-Projekte.

2.3.3 *Eignung für das pelican-Projekt*

Für das pelican-Projekt wurde aus folgenden Gründen ein serviceorientierter Ansatz gewählt:

1. **Die Projektstruktur.** Der SFB/TRR 77 unterteilt sich neben den Z-Projekten in 22 weitere medizinische Forschungsprojekte. Jedes davon hat eigene Anforderungen an die Software, benutzt größtenteils verschiedene Datenbasen und wendet unterschiedliche Auswertungs- und Analyseverfahren an. Nach traditionellem Ansatz müsste für jedes Projekt eine eigene Software entworfen und implementiert werden. Der Aufwand einer Integration mehrerer solcher Insellösungen steigt exponentiell. Mit SOA werden die Dienste(klassen) einmalig definiert und programmiert und können beliebig oft verknüpft und wiederverwendet werden.

2. **Vereinfachte Kooperation.** Mittels einer zentralen Dienstverwaltung wird es für die Projekte relativ einfach, Leistungsspektren der kooperierenden Projekte einzusehen und bereitgestellte Dienste in Anspruch zu nehmen. Als einfaches Beispiel ist folgendes Szenario denkbar:

   Projekt X stellt die Rohdaten seiner Microarray-Experimente auf der IT-Plattform als Datendienst zur Verfügung. Projekt Y hat eine konkrete Fragestellung, aber keine geeignete Datenbasis. Sie finden den

Dienst anhand einer Beschreibung auf der Plattform und können mit der Erlaubnis von Projekt X auf die Daten zugreifen und eigene Analysen durchführen. Die so gewonnenen Erkenntnisse können wiederum nützlich für Projekt X sein.

Im OASIS SOA-Referenzmodell wird dieses Prinzip als „matching of capabilities and needs“ beschrieben.

3. **Eigentümerschaft.** Die Wahrung der Hoheit über eigene Daten und Ergebnisse hat sich in Gesprächen mit den Forschern als zentrales Bedürfnis herausgestellt. Da das SOA-Paradigma ursprünglich für den Unternehmensbereich entwickelt wurde, trägt es der Forderung nach Kontrolle über die Eigentümerschaft in besonderem Maße Rechnung. Berechtigungs- und Sicherheitskonzepte lassen sich mit Werkzeugen aus dem SOA-Umfeld verhältnismäßig einfach umsetzen.

2.3.4 *Webservices*

Das Konzept der Serviceorientierten Architektur stellt nur ein Referenzmodell dar. Referenzmodell bedeutet, es handelt sich um ein allgemeines, idealtypisches Modell für eine Klasse von Sachverhalten und ihren Beziehungen zueinander. Davon ausgehend lassen sich spezielle, den konkreten Anwendungsfall betreffende Modelle ableiten. Für die spezifische Implementierung des SOA-Ansatzes eignet sich beispielsweise die so genannte Webservices-Technologie.

Das World Wide Web Consortium (W3C) als wichtigste internationale Organisation für die Etablierung von Standards im Umfeld des Internet definiert Webservices wie folgt:

> „A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [Booth et al. 2004].“

Hervorzuheben ist hierbei, dass der Benutzer typischerweise keinen Anteil am Informationsaustausch mittels Webservices hat, außer ihn gegebenenfalls zu initiieren. Das Clientprogramm (Dienstnutzer) kommuniziert automatisch über definierte Protokollstandards mit dem Dienstanbieter (zum Beispiel einem Webserver) und kann die Ergebnisse der Abfrage weiter verarbeiten. Anschaulich wird dieses Prinzip

in folgendem Beispielsszenario umgesetzt:

Der Kunde eines Reisebüros möchte einen Urlaubsflug buchen und gibt den gewünschten Reisezeitraum sowie weitere Parameter in die Suchmaske der Webseite des Reisebüros ein. Jetzt startet der Server mittels Webservices Anfragen an verschiedene Fluggesellschaften, die einen solchen Informationsdienst anbieten. Die Ergebnisse werden an die aufrufende Webseite zurückgeschickt und aufbereitet. Jetzt kann sich der Kunde für einen passenden Flug entscheiden und ihn gegebenenfalls direkt buchen. In den Datenaustausch zwischen der Internetseite des Reisebüros und den Servern der Fluggesellschaften war er nicht involviert.

In größerem Rahmen werden Webservices heutzutage in Form von weltumspannenden Netzwerken, so genannten Grids, eingesetzt.

2.3.5 *Grid Computing*

„The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations." So beschreibt Foster [2001] die Idee hinter dem Grid-Ansatz.

Der Begriff Grid (engl. Gitter, Raster) ist der englischen Bezeichnung des Stromnetzes (power grid) entlehnt. Der Nutzer soll, ähnlich der Benutzung einer Steckdose, Zugriff auf Ressourcen und Applikationen haben, ohne sich mit der Komplexität zugrundeliegender Systeme auseinandersetzen zu müssen. Er benötigt dazu nur einen funktionierenden Webbrowser, keine spezielle Software oder gar Wissen über Netzwerkeinstellungen [Thomas et al. 2005]. Foster zufolge ist das Grid aber nicht als Alternative zum Internet zu verstehen, sondern vielmehr als eine Sammlung zusätzlicher auf dem Internet aufbauender Protokolle und Dienste, um die Bereitstellung und Nutzung rechenintensiver und datenreicher Aufgaben durch Verteilung der Last zu unterstützen.

Ein Beispiel für ein solches Ressourcennetzwerk stellt das caBIG dar. Diese Initiative des amerikanischen National Cancer Institute (NCI) wurde im Jahr 2003 gestartet, um eine auf Standards basierende, biomedizinische Infrastruktur zu schaffen und diese allen Forschern auf dem Gebiet der Krebserkrankung uneingeschränkt zugänglich zu machen [Eschenbach und Buetow 2006]. Gerade in der Krebsforschung haben die Wissenschaftler mit unterschiedlichen Terminologien und dem Austausch heterogener Daten zu kämpfen. Durch den Zusammenschluss von Medizinern, Biologen, Bioinformatikern und Medizininformatikern zu einer gemeinnützigen Gemeinschaft sollen das Verständnis und die Behandlung der Krankheit maßgeblich verbessert

werden.

Die an Webservices orientierte Infrastruktur des caBIG Projekts heißt caGrid. Sie stellt Krebsforschern verschiedene Technologien zur Verfügung, um Daten und Analysewerkzeuge anzubieten, sicher austauschen und effektiv nutzen zu können. So kann beispielsweise die Durchführung koordinierter Krebsstudien mit mehreren beteiligten Institutionen verbessert werden [Saltz 2006].

Die pelican-Dienste sollen zu gegebener Zeit ebenfalls in das caGrid eingebunden werden. So kann das Projekt vom Fachwissen aller teilnehmenden Institutionen profitieren. Im besten Fall lassen sich nützliche Webservices in den Analyseprozess des Transregio 77 integrieren und so Implementierungsaufwand gespart werden. Im Gegenzug können im pelican-Projekt entwickelte Dienste von anderen caGrid Nutzern weltweit in Anspruch genommen werden und sind vielleicht einmal an einem bedeutenden Fortschritt in der Krebsforschung beteiligt.

2.4 SOFTWAREPORTALE

Die im SFB/TRR 77 beschäftigten Forscher haben jeweils individuelle Anforderungen an eine Software für ihre tägliche Arbeit. Diese werden in Kapitel 4.1 genauer erörtert. Ein mögliches Anwendungssystem (engl. Framework) für die Informationsplattform muss demnach hohe Flexibilität gewährleisten und einfach an persönliche Bedürfnisse anzupassen sein. Ähnlich wie das Konzept der Serviceorientierten Architektur haben sich so genannte IT-Portale als flexibel einsetzbare Frameworks zunächst aus wirtschaftlichen Interessen heraus entwickelt. Die darüber hinaus gehende Eignung für wissenschaftliche Zwecke, insbesondere Kollaborationsvorhaben, wird durch folgende Begriffserklärung deutlich.

2.4.1 *Definition und Eigenschaften*

Hinderer et al. [2004] definieren ein Portal als eine auf Webtechnologien basierende Applikation, welche einen zentralen Zugriff auf personalisierte Inhalte sowie bedarfsgerecht auf Prozesse bereitstellt. Charakteristisch ist, dass Portale verschiedenartige Anwendungen über eine Portalplattform mit homogener Benutzeroberfläche verknüpfen und den Datenaustausch zwischen ihnen ermöglichen. Sie bieten somit die Möglichkeit, Prozesse und Zusammenarbeit innerhalb heterogener Gruppen zu unterstützen.

Gemäß dieser Definition wird die Informationsplattform durch den Einsatz der Portaltechnologie optimal ergänzt. Die Teilprojekte des SFB/TRR können wegen der unterschiedlichen Fragestellungen, Analysemethodik und verwendeten Technologie (beziehungsweise Software) als heterogene Anwendergruppen angesehen werden. Integriert man nun die verschiedenen Anwendungen mittels Webservices und stellt diese auf einer einheitlichen Benutzeroberfläche zur Verfügung, können die Wissenschaftler in ihrer gemeinschaftlichen Forschungsarbeit sinnvoll unterstützt werden. Dabei sind folgende Eigenschaften hilfreich, die typischerweise mit einem Portal assoziiert werden:

- **Integration von Daten und Funktionen.** Datenquellen und Funktionalität aus mehreren Systemen und Anwendungen werden an das Portal angeschlossen und können über eine einheitliche Benutzeroberfläche angesprochen werden. Üblicherweise stellen Portale ihren Benutzern auch Anwendungen wie Blogs, Wikis und Dokumentenmanagementsysteme zur Verfügung.
- **Anpassung des Erscheinungsbildes (Customization).** Ein Benutzer kann das Aussehen der Benutzeroberfläche seinen persönlichen Vorlieben anpassen. Ebenso können Firmen ihren Un-

ternehmensportalen eine einheitliche Darstellung verleihen (das so genannte Corporate Design).

- **Anpassung an Bedürfnisse (Personalization).** Personalisierung bezieht sich eher auf die inhaltliche Komponente. Ein Benutzer stellt sich entsprechend seiner Bedürfnisse Dienste und Funktionen aus dem Gesamtangebot des Portals zusammen und kann sie daraufhin über die Benutzeroberfläche ansteuern.

- **Zugriffskontrolle.** Bezeichnet das Rollen- und Rechtemanagement eines Portals. Hier können diverse Einstellungen vorgenommen werden, wie die Beschränkung des Zugriffs auf Daten und Funktionen für bestimmte Nutzergruppen oder die Aufzeichnung von Nutzeraktivitäten in Protokolldateien.

- **Single Sign-On.** Nachdem sich der Benutzer im System angemeldet hat, kann er auf alle Funktionen des Portals zugreifen. Dies beinhaltet auch integrierte Anwendungen, für die sonst eine separate Anmeldung nötig wäre.

2.4.2 *Kategorien von Softwareportalen*

Web-Portale lassen sich gemäß ihrer Ausrichtung in verschiedene Kategorien unterteilen. Großmann und Koschek [2005] unterscheiden Portale nach folgenden Kriterien:

- **Fokus.** *Horizontale Portale* bieten dem Nutzer ein breit gefächertes Informationsangebot. *Vertikale Portale* spezialisieren oder beschränken sich auf bestimmte Themengebiete oder Funktionen. So haben sich beispielsweise Newsportale, Suchportale, Regierungsportale oder Börsenportale entwickelt.

- **Nutzerkreis.** *Offene Portale* sind öffentlich zugänglich und können, meist nach erfolgter Registrierung, von jedem Benutzer angesteuert werden. Im Gegensatz dazu stehen *geschlossene Portale* nur einer festgelegten Gruppe von Nutzern zur Verfügung. Dies erfordert die Implementierung eines entsprechenden Sicherheitskonzepts.

- **Rollen der Benutzer.** Die Beziehungen der Nutzer untereinander charakterisieren ein (Unternehmens-)Portal als *business-to-business*, *business-to-customer* oder *business-to-employee* Portal. Je nach Art und Zweck des Portals werden den Nutzern unterschiedliche Funktionen und Anwendungen angeboten.

- **Netzwerktechnische Erreichbarkeit.** Portale können in ein firmeninternes *Intranet* eingebettet sein, zusätzlich ausgewählten externen Nutzern zugänglich gemacht werden (*Extranet*) oder gänzlich über das *Internet* erreichbar sein.

Das Portal unserer Kollaborationsplattform pelican stellt nach dieser Typisierung ein vertikales, geschlossenes Portal dar, welches per Webbrowser über das Internet erreichbar ist.

2.4.3 *Architektur und Komponenten*

Nachdem bisher ausschließlich funktionelle Eigenschaften definiert wurden, widmet sich der folgende Abschnitt der technischen Realisierung eines Portals. Es ist sinnvoll, sich bei der Implementierung eines Portals an einer Referenzarchitektur zu orientieren. Gurzki und Hinderer [2003] beschreiben in ihrer Referenzarchitektur für Unternehmensportale, wie Portalkomponenten klassischerweise interagieren (siehe Abbildung 2.3). Zunächst lassen sich Portale entsprechend dem Model-View-Control-Entwurfsmuster der Softwareentwicklung nach Gamma [2003] in drei Bereiche beziehungsweise Schichten unterteilen:

Das *Datenmodell* oder auch Backend-Schicht stellt Datenquellen oder externe Anwendungen bereit. In der pelican-Anwendung sind dies die Datenbanken der wissenschaftlichen Projekte, welche beispielsweise klinische Daten, Arrayexperimente oder Dokumentensammlungen enthalten können.

In der *Controller-Schicht*, oder Anwendungslogik, werden Dienste, Funktionen und Benutzereingaben verwaltet und auf Systemereignisse mit der Ausführung entsprechender Prozesse reagiert. Dies geschieht bei einer Portalsoftware durch bestimmte Basisdienste wie die Rechte- und Benutzerverwaltung.

Die *Präsentationsschicht* ist ausschließlich für die Darstellung der Daten und die Entgegennahme von Benutzereingaben zuständig. Typischerweise übernimmt diese Aufgabe ein Webbrowser. Im Fall der Informationsplattform kommen darüber hinaus spezielle Oberflächenkomponenten zur Anwendung, so genannte Portlets.

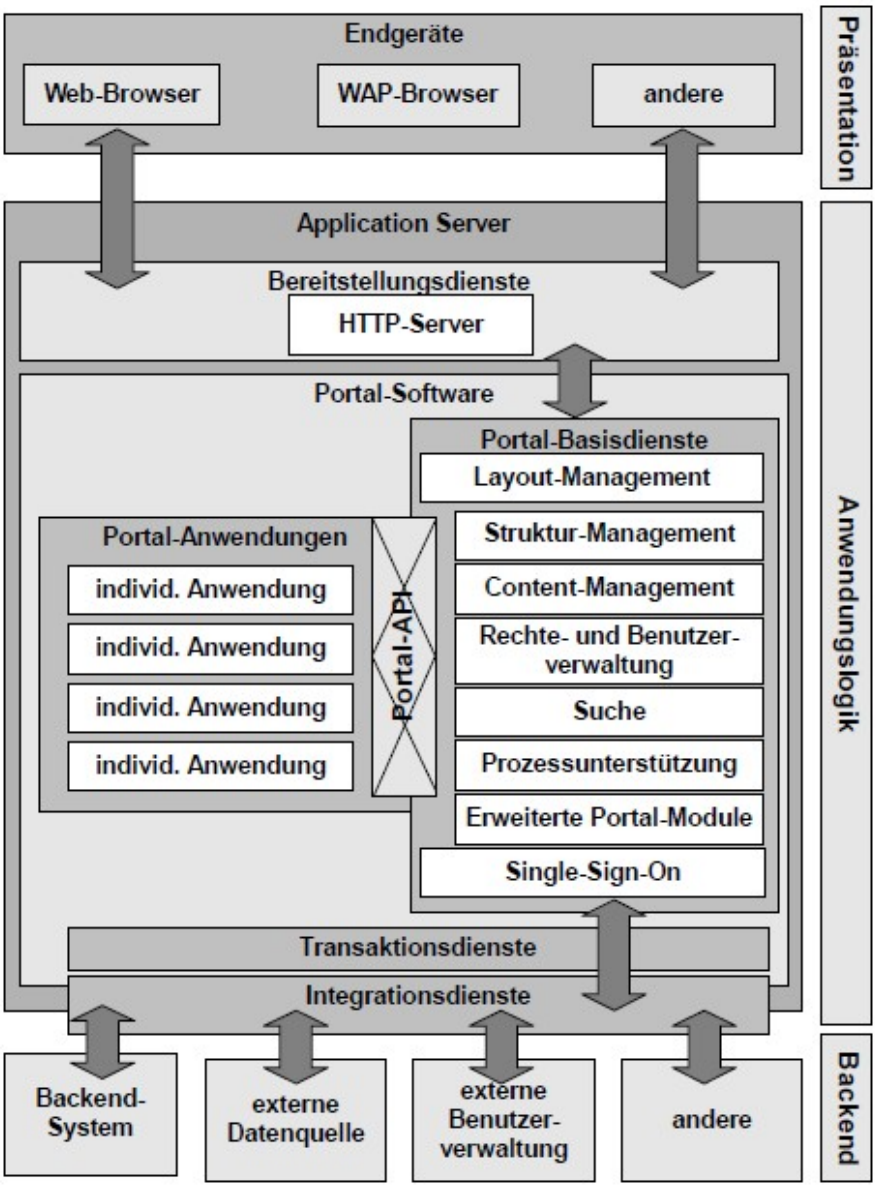

Abbildung 2.3: Referenzarchitektur für Softwareportale nach [Gurzki und Hinderer 2003].

2.4.4 *Portlets*

Die Java Portlet Specification Version 1.0 aus dem Jahr 2003 definiert Portlets folgendermaßen:

> „A portlet is a Java technology based web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems [Java Portlet Specification Version 1.0, 2003]."

Diese Komponenten der Benutzeroberfläche dienen demnach der Darstellung von Anwendungsinhalten, Diensten oder Daten (vgl. Abbildung 2.4). Verschiedene Portlets können beliebig kombiniert werden und tragen damit wesentlich zur Personalisierbarkeit der Portaloberfläche bei.

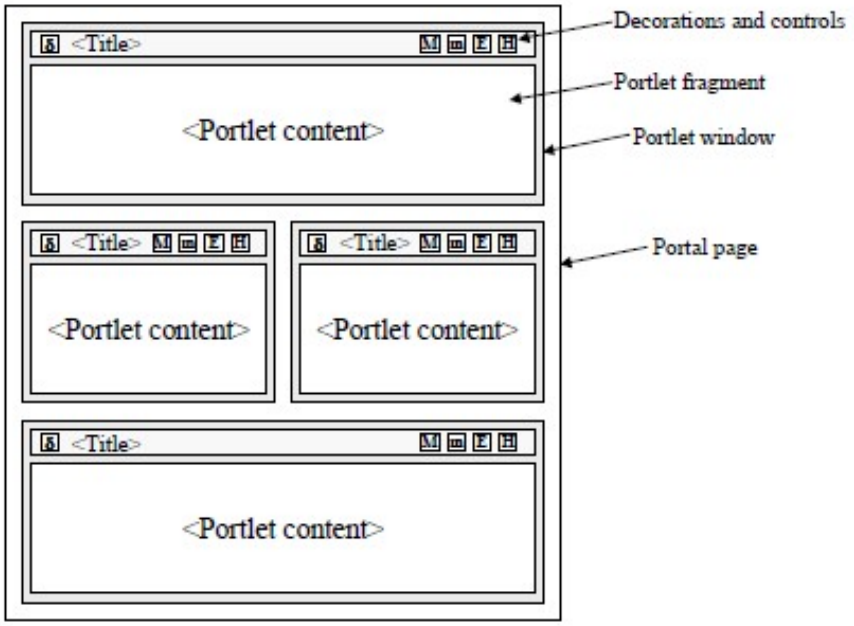

Abbildung 2.4: Schematische Benutzeroberfläche eines Portals mit Portlets gemäß der Java Portlet Specification JSR 168.

2.4.5 *Standards*

Die Einhaltung von allgemein akzeptierten und verbreiteten Standards in der Softwareentwicklung gewährleistet eine Wiederverwendbarkeit und Interoperabilität der Komponenten untereinander und über Systemgrenzen hinweg. Dabei reichen die Inhalte von reinen Übertragungsstandards bis hin zu konkreten Schnittstellenbeschreibungen. Insbesondere im Bereich der Softwareportale haben sich etliche Standards etabliert, von denen hier nur die wichtigsten erwähnt werden sollen. Die Auflistung orientiert sich an Kapitel 7 von Großmann und Koschek [2005].

- Präsentation und Design eines webbasierten Portals
  - **Extensible Markup Language (XML).** Der Metastandard zum Austausch von Daten zwischen unterschiedlichen Anwendungen ist Basis vieler moderner IT-Systeme.
  - **Hypertext Markup Language (HTML).** Dient der Strukturierung von Inhalten von Webseiten und Dokumenten. Dies

können beispielsweise Texte, Bilder oder Links zu anderen Seiten sein.

  - **Cascading Style Sheets (CSS).** Während mit HTML nur die inhaltliche Struktur eines Dokuments festgelegt werden sollte, kann mit CSS die konkrete Darstellung und Formatierung angepasst werden (beispielsweise Farbe, Schriftart und Größe einer Überschrift).

- Integration von Anwendungen
  - **Webservices.** Bereits in Kapitel 2.3 beschrieben, dienen Webservices dem Datenaustausch von Softwareanwendungen. Sie sind durch einen Uniform Resource Identifier (URI) eindeutig identifizierbar und besitzen eine in XML definierte Schnittstelle.
  - **Lightweight Directory Access Protocol (LDAP).** Ein Verzeichnisdienst (engl. directory service) bezeichnet eine durchsuchbare Datenbank, die autorisierten Nutzern oder Diensten Zugriff auf Informationen zu Personen, Rechnern, Netzwerkgeräten und Applikationen gewährt [Koutsonikola und Vakali 2004]. Das Anwendungsprotokoll LDAP ist ein offenener Industriestandard, der diesen Zugriff über IP-Netzwerke regelt. Im pelican-Projekt ist eine Installation des OpenLDAP Servers im Einsatz, um benutzerspezifische Informationen wie Passwörter und Gruppenzugehörigkeit vorzuhalten.

- Portaltechnik
  - **Portlets.** Die Java Portlet Specifications JSR168 und JSR286 der Java Community Process Gruppe (JCP) ermöglichen eine Interoperabilität von Portlets aus verschiedenen Webportalen, insbesondere die Personalisierung, Darstellung und Sicherheit betreffend. Während JSR168 die Programmierschnittstelle von Portlet und Container definiert, ergänzt der Nachfolger JSR286 die Spezifikation um Standards zur Inter-Portlet-Kommunikation und Einbindung externer Ressourcen.
  - **Web Services for Remote Portlets (WSRP).** Dieses Protokoll des Herstellerkonsortiums OASIS definiert Schnittstellen für den Fernzugriff auf interaktive, präsentationsorientierte Dienste, die auf externen Systemen zur Verfügung gestellt werden (beispielsweise als Portlets).

- Portalinhalte
  - **Really Simple Syndication (RSS).** Damit wird eine Sammlung von XML-Formaten bezeichnet, die Nachrichtenkanäle

(Channels) und Informationsobjekte (Items) definieren. Diese Objekte besitzen mindestens einen Titel, einen Verweis auf die eigentliche Nachricht und eine Beschreibung des Inhalts. Neuere Versionen von RSS können darüber hinaus noch weitere Informationen enthalten.

Bei allen vorgestellten Standards außer RSS handelt es sich um so genannte offene Standards. Im Gegensatz zu proprietären Standards sind sie öffentlich, kostenlos und herstellerunabhängig verfügbar.

2.4.6 *Portalsoftware*

Als Portalsoftware bezeichnet man Applikationen, die Unternehmen oder Nutzergruppen den Aufbau von IT-Portalen ermöglichen. Sie stellen die typischen Komponenten eines Portals als Paket zur Verfügung und müssen vom Portalbetreiber nur noch auf die jeweiligen Bedürfnisse und die verwendete IT-Infrastruktur angepasst werden. Auch hier kann man zwischen kommerziellen Anbietern (wie beispielsweise das WebSphere Portal der Firma IBM oder die Software SAP NetWeaver) und lizenzkostenfreien Produkten (zum Beispiel OpenSAGA oder GridSphere) unterscheiden. Darüber hinaus variieren die Portallösungen in Funktionsumfang und verwendeten Webtechnologien.

Aufgabe der vorliegenden Arbeit ist es unter anderem, die Anforderungen an eine Portallösung für den SFB/TRR 77 zu ermitteln, eine geeignete Portalsoftware auszuwählen und die pelican-Portallösung damit prototypisch zu implementieren.

3 METHODIK

Formuliertes Ziel der vorliegenden Diplomarbeit ist die Entwicklung einer prototypischen Benutzeroberfläche für die pelican-Anwendung.

Diese Oberfläche soll nicht von Grund auf neu programmiert, sondern mit Hilfe einer Portalsoftware (vgl. Kapitel 2.4.6) realisiert werden. Damit eine konkrete Softwarelösung ausgewählt werden kann, muss zunächst eine systematische Anforderungsanalyse durchgeführt werden. Als Ergebnis der Analyse wird ein Kriterienkatalog erarbeitet, der relevante Eigenschaften von Portalsoftware technischer und funktioneller Natur erfasst. Auf Basis dieser Übersicht werden verschiedene gängige Portallösungen miteinander verglichen und eine Auswahl für die Verwendung auf der Informationsplattform getroffen.

Für die Implementierung muss die am ehesten geeignete Portalsoftware an ein zuvor erstelltes Oberflächenkonzept angepasst werden. In diesem Konzept werden prinzipielle Designfragen wie Inhalt und Anzeige der Module sowie die Navigation zwischen Oberflächenkomponenten festgehalten.

Um die Eignung der entwickelten Benutzeroberfläche in der Praxis zu belegen, sollen bereits entwickelte Webservice Prototypen des pelican-Projekts als funktionale Bausteine, so genannte Portlets, integriert werden.

Zur Entwicklung von Portlets für die pelican-Anwendung wird die Entwicklungsumgebung (engl. integrated development environment (IDE)) Eclipse Version 3.6 *Helios* der Eclipse Software Foundation benutzt. Die ursprünglich für die Entwicklung von Java-Programmen konzipierte IDE ist dank ihrer zahlreichen quelloffenen Module, auch Plugins genannt, flexibel einsetzbar. Ausschlaggebend waren neben bisherigen positiven Erfahrungen mit Eclipse die zahlreich verfügbaren Anleitungen zur Integration von GridSphere in die Entwicklungsumgebung.

4 ERGEBNISSE

Dieses Kapitel widmet sich zunächst der Evaluation und Auswahl einer geeigneten Portalsoftware für den Einsatz innerhalb der pelican-Anwendung. Im zweiten Abschnitt wird ein Oberflächenkonzept erarbeitet und mit Hilfe der bereitgestellten Oberflächenkomponenten der Portallösung umgesetzt. Eine Validierung des generischen Portletansatzes erfolgt durch die prototypische Integration bereits entwickelter pelican-Module. Dabei überwundene spezifische Herausforderungen werden für zukünftige Weiterentwicklungen dokumentiert.

4.1 EVALUATION POTENZIELLER PORTALLÖSUNGEN

Portalsoftware stellt eine Möglichkeit dar, die vielen nützlichen Komponenten eines Portals als Paket herunterzuladen, zu installieren und den eigenen Vorstellungen entsprechend zu konfigurieren. Die Produkte verschiedener Hersteller unterscheiden sich neben weiteren Aspekten im Funktionsumfang, den Möglichkeiten der Anpassung und den verwendeten Basistechnologien. Es ist deshalb notwendig, die Anforderungen der pelican-Anwendung zu formulieren und eine Evaluation ausgewählter Portallösungen durchzuführen.

4.1.1 *Reviews*

Leider lassen sich in der Literatur nur wenige Evaluationsstudien zu Portalsoftware finden. Speziell Open Source Lösungen, auf deren Charakteristika zu einem späteren Zeitpunkt noch eingegangen wird, sind selten Thema vergleichender Analysen. Zudem stammt derartige Literatur oftmals aus den Anfangsjahren der Portaltechnologie. Wie für das schnelllebige IT-Umfeld üblich, haben sich die Kandidaten seitdem teilweise erheblich weiterentwickelt.

In einer Studie von Akram et al. [2005] wurden die Kernfunktionen von vier Portalframeworks identifiziert und in Kategorien eingeteilt. Die Auswahl der Kandidaten geschah hierbei auf Grund von allgemeiner Verbreitung und persönlichen Erfahrungen. Für jeden Kandidaten wurden je nach Erfüllungsgrad 1 bis 5 Punkte pro Kategorie vergeben und am Ende aufsummiert, wobei das Portal Liferay die höchste Punktzahl erreichte. Dieses Vorgehen lässt jedoch Raum für Diskussionen, denn die Vergabe der Punkte wurde nicht transparent dargestellt und berücksichtigt ebenso wenig die unterschiedliche Relevanz der

Funktionen.

Browning [2003] wählte in seiner Studie einen ähnlichen Ansatz, allerdings wurden andere Produkte verglichen. Im Gegensatz zu Akram et al. wurden die Kriterien in Subkategorien unterteilt, denen jeweils ein spezifisches Gewicht zugeordnet wurde. Auch hier wurden Punkte (nach einem nicht näher erläuterten System) vergeben und am Ende aufsummiert. Die höchste Punktzahl erhielt die Software uPortal.

Ein Ansatz, der in abgewandelter Form bei der Evaluation möglicher Portalframeworks in dieser Arbeit verfolgt werden soll, wurde von Dion Goh et al. [2008] gewählt. In dieser Studie wurden vier Kandidaten ausgewählt, die sowohl eine gewisse Anzahl an Installationen vorzuweisen haben als auch in mindestens einer Studie evaluiert worden sind. Den Kategorien Taxonomie, Metadaten, Suche, Anpassung und Personalisierung, Zusammenarbeit und Kommunikation, Sicherheit und Interoperabilität wurden verschiedene Gewichtungen zugeordnet. In einer Checkliste wurden die Fragen zu den einzelnen Kategorien unabhängig von drei Experten mit „nicht vorhanden“ (0 Punkte), „teilweise unterstützt“ (0.5 Punkte) und „unterstützt“ (1 Punkt) beantwortet und die Punktwerte mit den Gewichten (fünffach bis einfach) multipliziert.

Dieses Vorgehen ermöglicht die Erstellung eines genauen Profils jeder Portallösung durch das Aufzeigen individueller Stärken und Schwächen. Der absolute Sieger nach Punkten muss keinesfalls immer die beste Lösung für eine spezifische Plattform anbieten. Negativ ist zu vermerken, dass gerade im wichtigen Komplex „Interoperabilität“ viele Technologien aufgeführt wurden, die völlig unterschiedliche Hintergründe besitzen und im engeren Sinne nicht auf Interoperabilität ausgelegt sind (beispielsweise Java Database Connectivity JDBC). Eine präzisere Überschrift zu den Fragen dieser Kategorie wäre Standardkonformität gewesen.

4.1.2 *Checklisten als Evaluationswerkzeuge*

Die Verwendung von Checklisten ist im Umfeld der Softwareentwicklung weit verbreitet. Ihre Eignung für Evaluationszwecke wurde bereits hinreichend belegt, zum Beispiel in einer Studie von Stufflebeam [2001]. Ihre Anwendung ist einfach: Der Benutzer kann mittels einer systematisch strukturierten Liste entscheiden, ob und in welchem Grad der Untersuchungsgegenstand gegebene Kriterien erfüllt. Die Kriterien werden häufig als Fragen formuliert. Die Checkliste kann sowohl quantitative (zum Beispiel nach den Anschaffungskosten) als auch qualitative Fragen (wie etwa die empfundene Einfachheit der Installation) enthalten. Bei Fragen qualitativer Natur bietet sich zum

Zwecke der Vergleichbarkeit eine Einteilung der Antwort in Skalen an.

In Anlehnung an Dion Goh et al. soll im Folgenden das gleiche Wertungssystem für die Unterstützung von Funktionen benutzt werden: Eine Wertung von 0 Punkten bedeutet, dass die Software die Funktion nicht besitzt oder nicht unterstützt. Im Gegensatz dazu wird 1 Punkt vergeben, wenn die Funktion vorhanden beziehungsweise voll unterstützt wird. Kann die Funktionalität nur durch zusätzlichen Programmieraufwand oder nicht triviale Anpassungen erreicht werden, werden 0.5 Punkte verliehen. Für die Gewichtung der Kategorien wird in dieser Arbeit allerdings nur eine Skala mit drei Werten genutzt:

- einfache Gewichtung für rein ästhetische Funktionen
- doppelte Gewichtung für sekundäre Funktionen (sprich, im pelican-Kontext nebensächlich)
- dreifache Gewichtung für Kernfunktionalität, welche die Ziele unserer Informationsplattform maßgeblich unterstützt.

Nachdem für jede Kategorie Punktwerte vergeben und mit der Gewichtung multipliziert worden sind, lassen sich die Kandidaten über die aufsummierte Gesamtwertung vergleichen. Es kann der Fall eintreten, dass die Wertungen eine geringe Varianz aufweisen. Das ist problematisch, wenn diese Äquivalenz der Produkte nur scheinbar besteht. Die Kandidaten sind also nicht auf allen Gebieten ähnlich gut oder schlecht, sondern Software A brilliert in einer Kategorie, in der Software B nur wenige Punkte erhielt und umgekehrt. In diesem Fall sind Prioritäten zu setzen.

Für die Beantwortung der Fragen soll für alle Kandidaten eine Testinstallation aufgesetzt werden, in welcher die einzelnen Kriterien, soweit möglich, überprüft werden können.

4.1.3 *Kategorien*

Die erstellte Anforderungs-Checkliste (siehe Anhang B) orientiert sich an den bereits genannten Kategorien von Dion Goh et al. Jedoch besitzen nicht alle genannten eine spezifische Relevanz für das pelican-Projekt. Die Themenkomplexe Taxonomien, Metadaten und Content Management (beinhaltet Suche, Dokumentenaustausch und Werkzeuge zur Zusammenarbeit) werden bereits durch andere Applikationen innerhalb der Plattform abgedeckt und fließen nicht in die Bewertung ein. Stattdessen wird die Matrix um zwei zusätzliche Kategorien (Support und Besonderheiten) erweitert.

Offizielle Arbeitssprache innerhalb des TRR ist auf Grund des internationalen Umfelds Englisch. Eine Unterstützung zur Lokalisierung des Portals für weitere Sprachen ist vorerst nicht vorgesehen und findet deshalb keine Beachtung bei der Bewertung.

Im Folgenden werden die zur Evaluation genutzten Kategorien charakterisiert. Bei der Einteilung wurde darauf geachtet, dass keine Überschneidungen zwischen den Kategorien existieren und somit die Gewichtung der Punkte verzerrt würde.

Technische Rahmenbedingungen

Umgebung. Die Portalsoftware muss für die Installation auf dem TRR-Server ausgelegt sein, der unter dem Betriebssystem Linux OpenSuse Version 11 läuft.

Standardkonformität. Die Idee bei der Verwendung von Standards, namentlich die Gewährleistung von Interoperabilität und Wiederverwendbarkeit von Systemkomponenten, wurde bereits in Kapitel 2.4.5 diskutiert. In die Bewertung fließt die Unterstützung der Standards JSR 168 und JSR 286 sowie des LDAP Protokolls ein.

Datenbankanbindung. Einige Portallösungen benutzen für die Speicherung interner Daten integrierte Datenbanken. Bei anderen muss ein Datenbankmanagementsystem (DBMS) bereits auf dem Zielsystem installiert sein. Optimal wäre die Unterstützung möglichst vieler gängiger kostenloser DBMS wie MySQL und postgreSQL. In diesem Fall kann der Administrator dasjenige auswählen, welches den technischen Anforderungen am ehesten gerecht wird.

Gridfunktionalität. Da die pelican-Anwendung zukünftig in den caGrid Kontext eingebunden werden soll (siehe Kapitel 2.3.5), erspart eine Unterstützung von Gridfunktionalität unter Umständen viel Entwicklungsaufwand.

Support

Eine umfangreiche und verständliche Dokumentation sowie Installationsleitfäden und Supportforen sind große Pluspunkte bei der Auswahl einer geeigneten Portalsoftware. Darüber hinaus kann dem IT-Mitarbeiter in Form von zusätzlichen Programmierschnittstellen, so genannten Application Programming Interfaces (APIs), die Entwicklung neuer Funktionen und Komponenten erleichtert werden.

Layoutmanagement

Layoutmanagement ist eine Form des Informationsmanagements. In Anlehnung an Haux [2004] besteht das Ziel von Informationsmana-

gement darin, die korrekte Information zum richtigen Zeitpunkt am richtigen Ort der berechtigten Person in einer geeigneten Form zur Verfügung zu stellen. Es liegt jedoch in der Natur von Kollaborationsplattformen, dass Informationen aus verschiedenen Quellen stammen und einen uneinheitlichen Grad an Relevanz und Strukturierung aufweisen. Die Portalsoftware muss nun entsprechende Mechanismen zur Verfügung stellen, um heterogene Informationen und Daten zu kapseln und dem Benutzer auf einer einheitlichen Benutzeroberfläche zu präsentieren.

Eine optimale Portaloberfläche zeichnet sich unter anderem durch die übersichtliche Darstellung aller relevanten Informationen für einen bestimmten Nutzer aus. Diese Eigenschaft unterliegt jedoch einer gewissen Subjektivität, bedingt durch unterschiedliche Nutzerrollen, Fähigkeiten und Vorlieben. Die Möglichkeit der Customisierung (von engl. to customize = anpassen, individualisieren), also einer Anpassung an eigene Präferenzen und Bedürfnisse, erhöht die Benutzerfreundlichkeit und infolgedessen die Akzeptanz des Portals durch den Nutzer. Dies können zum Beispiel verschiebbare Elemente, spezielle Farbschemata oder ein- und ausblendbare Schaltflächen sein.

Einmal vorgenommene Einstellungen und ausgewählte Inhalte beziehungsweise Funktionen sollen dauerhaft in einem Benutzerprofil gespeichert werden können und beim nächsten Programmstart zur Verfügung stehen. Diese Funktion bezeichnet man als Personalisierung.

Sicherheit

In Gesprächen mit Projektmitarbeitern wurde stets die große Bedeutung der Hoheit über eigene Daten hervorgehoben. Innerhalb des TRR soll deshalb ein Gremium (Access Authorization Board, AAB) gewisse Regeln zum Umgang mit Daten aus anderen Teilprojekten und die Freigabe von eigenen, womöglich noch nicht publizierten Ergebnissen festlegen. Die Portalsoftware muss in der Lage sein, ein solches Datenschutzkonzept umzusetzen. Folgende Funktionen spielen im Sicherheitskontext der Applikation eine Rolle.

Single Sign-On. Mit Single Sign-On (engl. einmaliges Anmelden) wird zum Ausdruck gebracht, dass in der gesamten Systemlandschaft nur eine einmalige Authentifizierung erforderlich ist, unabhängig von gegebenenfalls integrierten heterogenen Anwendungen. Abbildung 4.1 verdeutlicht das Prinzip. Anstatt für jedes Anwendungssystem eigene Authentifizierungsprozesse zu starten und jedes Mal Passwörter und Benutzernamen abzufragen, werden Benutzerdaten zentral verwaltet und entsprechende Informationen zwischen den Softwarekomponenten ausgetauscht. Im Fall der pelican-Anwendung übernimmt dieses

Rechte- und Rollenmanagement ein spezieller LDAP-Server, auf dem auch alle Benutzerdaten hinterlegt sind. Dadurch wird die Administration der Login-Daten wesentlich erleichtert.

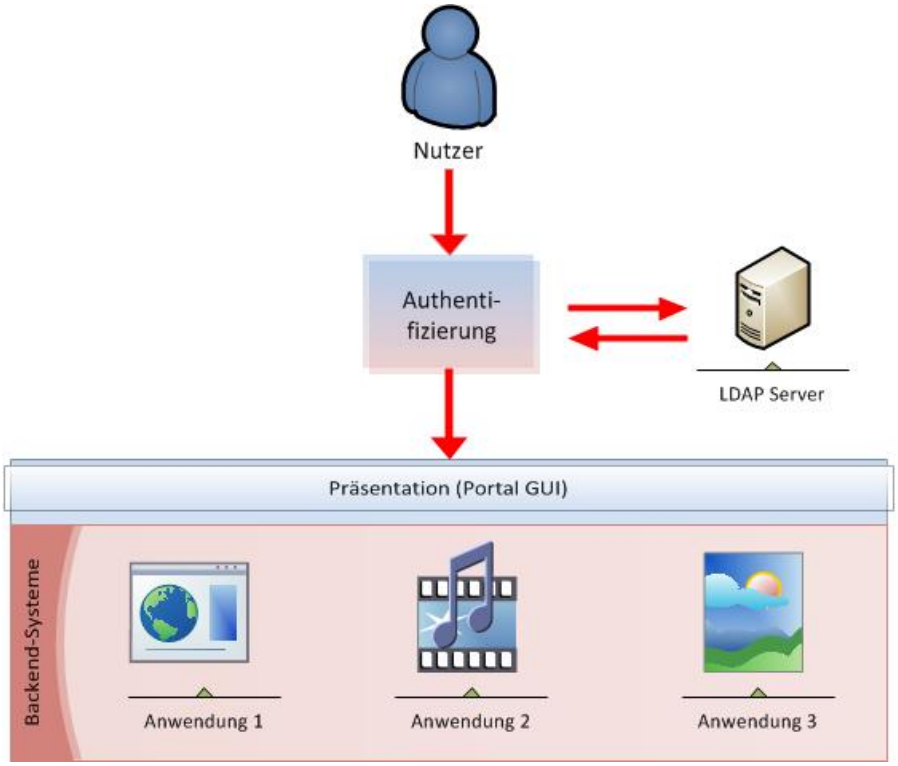

Abbildung 4.1: Anmeldungsprozess mittels Single Sign-On, den jeder Nutzer einmalig pro Sitzung durchläuft. In Abhängigkeit von seiner Rolle und die mit seinem Benutzerkonto verbundenen Berechtigungen erhält er Zugriff auf Ressourcen und Funktionen.

Gruppen- und Rechtemanagement. Eng verknüpft mit den Themen Datenschutz und Single Sign-On ist eine Zuweisung von definierten Benutzergruppen mit bestimmten Rechten zu Benutzerprofilen. Ein Benutzer kann dabei mehreren Gruppen angehören. Im Beispiel des TRR könnte eine Person gleichzeitig „Projektgruppenleiter“ als auch „Datenschutzbeauftragter“ sein. Seine Rechte in der Applikation ergeben sich als Summe derer, die den einzelnen Gruppen zugewiesen sind. Dementsprechend kann auch die Benutzeroberfläche rollenspezifisch angepasst werden und jeweils benötigte Funktionen zur Verfügung stellen. Die Portalsoftware sollte ein zentrales Werkzeug zur Verwaltung der Gruppenprofile und Rechtevergabe bereitstellen.

Logging. Um Nachvollziehbarkeit und Integrität der Daten jederzeit gewährleisten zu können, sollten Protokollmechanismen vorhanden

sein, die Nutzeraktionen erfassen, Systemereignisse registrieren und bei Bedarf in geeigneter Form darstellen können.

Besonderheiten

Diese Kategorie soll Portallösungen die Möglichkeit geben, über für das pelican-Projekt relevante Sonderfunktionen zusätzliche Punkte zu erreichen. Das könnten zum Beispiel Metadatenfunktionalität, enthaltene Statistikwerkzeuge oder besondere Sicherheitsmechanismen und Zertifikate sein.

4.1.4 *Kandidaten*

Allein im Bereich der auf Java basierenden Open Source Portalsoftware führte die Internetseite `java-source.net`[1] zweiundzwanzig Softwarelösungen auf.

Als Open Source Software (OSS) bezeichnet man quelloffene Programme, deren Verbreitung und Modifikation im Allgemeinen nicht beschränkt ist. Die offizielle Definition der Open Source Initiative[2] geht weit darüber hinaus. Im Rahmen dieser Evaluation ist jedoch nur wichtig, dass der Quellcode unseren Bedürfnissen entsprechend angepasst werden darf. Darüber hinaus tendiert OSS im Gegensatz zu kommerziellen Produkten zu einer rascheren Weiterentwicklung, besserer Unterstützung von Standardtechnologien und effizienterer Fehlerbehebung durch eine aktive Nutzergemeinschaft [McAllister 2004].

Wegen des begrenzten Budgets für IT-Infrastruktur, den enormen Kosten kommerzieller Lösungen und einer Vielzahl an Open Source Alternativen beschränkt sich die Evaluation der Portalsoftwares für die Verwendung im pelican-Projekt auf lizenzkostenfreie Produkte.

GridSphere

Zu Projektbeginn im Jahr 2002 war selbsterklärtes Ziel des EU geförderten GridLab Projekts die Schaffung des besten, alle gängigen Standards unterstützenden Open Source Portals. Dabei sollten auch Ansätze der Grid Technologie verfolgt werden.

Im November 2010 wurde GridSphere Version 3.2 veröffentlicht. Die Portalsoftware ist mittlerweile bei vielen, vor allem wissenschaftlichen Portalen im Einsatz, beispielsweise im MediGRID Projekt oder innerhalb des Biomedical Informatics Research Network (BIRN). Das Framework liefert mit einer Anzahl von Kernkomponenten (unter anderem fertige Portlets für Login, Logout, Lokalisierung, Account

1 `http://java-source.net/open-source/portals`, zuletzt abgerufen am 03.02.2011
2 `http://www.opensource.org/docs/osd`, zuletzt abgerufen am 28.12.2010

Management, Benutzermanagement und Layout Konfiguration) die Basisfunktionalität eines Portals. Für die erwähnte Gridkompatibilität wird allerdings eine spezielle Webapplikation namens GridSphere Grid Portlets benötigt; sie ist im Framework nicht enthalten.

Neben dem Fokus auf wissenschaftliche Kollaborationsportale (vgl. Referenzen in Anhang B) ist die postulierte Gridfunktionalität dafür ausschlaggebend, GridSphere in die Evaluation einzubeziehen. Die Testinstallation (siehe Abbildung 4.2) verlief dank der umfangreichen Dokumentation und Benutzerfreundlichkeit des Installationspakets einfach und schnell. Lediglich die notwendigen Programme Java und Ant mussten sich auf dem System befinden und korrekt konfiguriert werden.

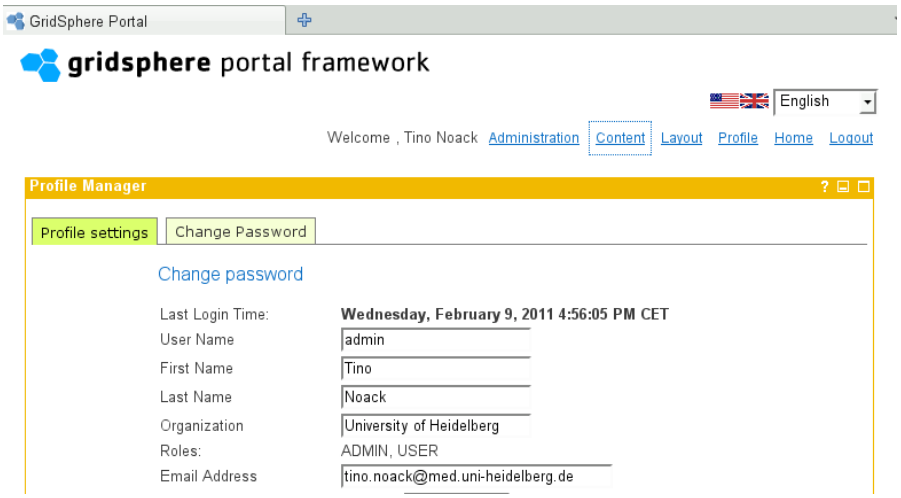

Abbildung 4.2: Screenshot der GridSphere Testinstallation.

LifeRay

Seinen Ursprung hat die Open Source Portalsoftware Liferay im Jahr 2000. Um den wachsenden Ansprüchen der Nutzergemeinschaft zu genügen, wurde 2004 die Firma Liferay, Inc. gegründet. Seitdem hat sich das Unternehmen mit seiner Portallösung neben großen, kommerziellen Konkurrenten auf dem Markt etabliert. Es wurde 2010 als einziges Open Source Portal als einer der Marktführer im renommierten „Magischen Quadranten der horizontalen Portale“ (siehe Abbildung 4.3) vom Marktforschungsunternehmen Gartner aufgeführt und soll deshalb im Rahmen der Evaluation genauer betrachtet werden.

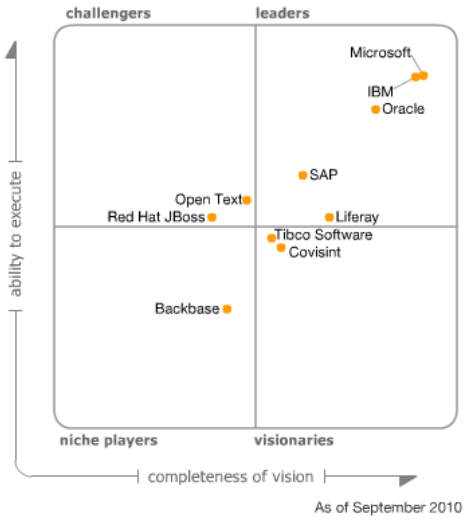

Abbildung 4.3: „Magischer Quadrant der horizontalen Portale“ des Marktforschungsunternehmens Gartner, 2010.[3]

Mittlerweile ist Liferay in der Version 6.0 verfügbar. Neue Versionen werden in regelmäßigen Abständen veröffentlicht. Das Unternehmen gibt auf seiner Internetseite Nutzerzahlen an die 3 Millionen an, mit 250.000 Installationen weltweit. Unter den Nutzern befinden sich bedeutende Firmen wie Cisco und Honda, aber auch das bereits angesprochene caGrid basiert auf Liferay Technologie.

Die Installation einer Testversion (vgl. Abbildung 4.4) verlief dank eines komfortablen Installationspakets sogar noch einfacher als bei GridSphere, lediglich ein Speichergrößenproblem unter Java sorgte für Komplikationen. Dieses konnte mit Hilfe von Beiträgen im Nutzerforum schnell behoben werden.

3 `http://www.gartner.com/technology/media-products/reprints/liferay/206214.html`, zuletzt abgerufen am 22.12.2010

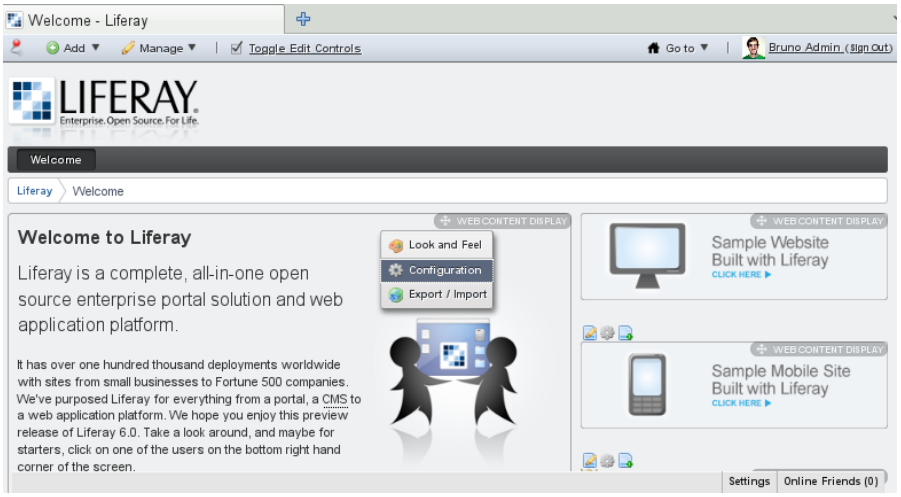

Abbildung 4.4: Screenshot der Liferay Testinstallation.

GateIn Portal

GateIn Portal ist das Ergebnis vom Zusammenschluss zweier etablierter Hersteller auf dem Open Source Portalmarkt, namentlich JBoss Portal und der eXo Platform. Durch diesen Schritt im Juni des Jahres 2009 sollten die jeweiligen Expertisen für ein gemeinsames Produkt einfließen. Während die Stärken von JBoss in bei der Integration von Komponenten und Modularität liegen, ist eXo zuständig für die Entwicklung der Benutzeroberfläche und Administrationswerkzeuge.

Sowohl JBoss Portal als auch die eXo Platform wurden bereits in Evaluationsstudien betrachtet (Akram et al., Dion Goh et al.). Die Evaluationen fanden allerdings vor dem Zusammenschluss der Projekte statt. Deshalb ist die kombinierte Funktionalität des GateIn Portals eine erneute Betrachtung in Form einer Testinstallation (Abbildung 4.5) wert.

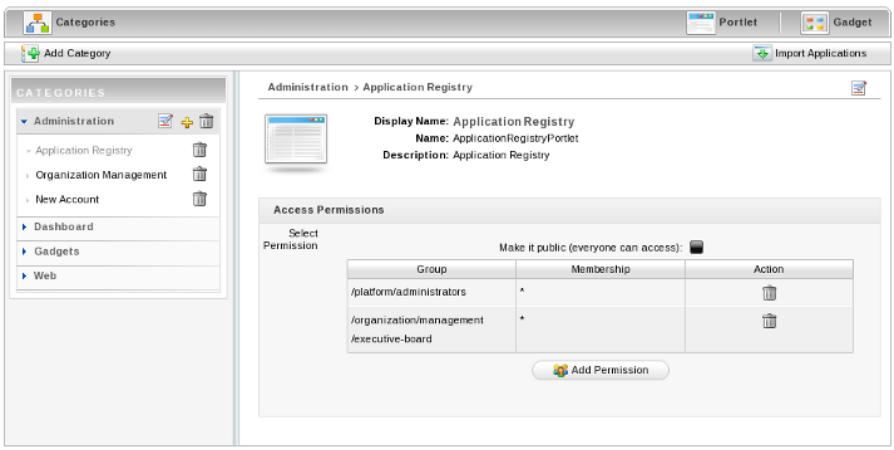

Abbildung 4.5: Screenshot der GateIn Testinstallation.

Tabelle 4.1: Ergebnisse der Evaluation (Kurzform).

	GRIDSPHERE	LIFERAY	GATEIN PORTAL
Techn. Rahmenbedingungen (max. 15)	13.5	12	12
Support (max. 6)	5	5	4
Layoutmanagement (max. 4)	3	4	4
Sicherheit (max. 15)	13.5	12	10.5
Besonderheiten (max. 5)	0	2	2
Gesamtwertung (max. 45)	35	35	32.5

4.1.5 *Ergebnisse*

Tabelle 4.1 lassen sich die Ergebnisse der Checklistenevaluation für die drei Portale entnehmen. Im Folgenden soll das Abschneiden der einzelnen Kandidaten in den unterschiedlichen Kategorien verglichen werden.

Technische Rahmenbedingungen
Alle drei Kandidaten erfüllen die in Kapitel 4.1.3 formulierten technischen Anforderungen in nahezu gleichem Maße. Lediglich die Unterstützung von Gridfunktionalität ist GridSphere vorbehalten, wenn auch nur in Ansätzen.

Support
Die Unterstützung des Programmierers seitens der Software ist durchweg gut. Die Leitfäden beschreiben Abläufe wie Installation oder Konfiguration detailliert. Eine zusätzliche Programmierschnittstelle konnte aus technischen Gründen nicht für jedes Produkt identifiziert werden.

Layoutmanagement
Das Aussehen jeder Portallösung ließ sich an persönliche Vorlieben anpassen, sowohl für die gesamte Applikation als auch benutzerspezifisch. Im Vergleich zu den beiden Konkurrenten wirkt die Standardoberfläche von GridSphere ohne zusätzlichen Programmieraufwand jedoch etwas altbacken und ist weniger komfortabel zu konfigurieren,

beispielsweise wegen fehlender Drag-and-Drop[4] Funktionalität.

Sicherheit
Die meisten Punkte in dieser Kategorie konnte GridSphere erreichen. Der geringe Vorsprung resultiert aber lediglich aus der teilweise vorhandenen Loggingfunktion, die die Konkurrenz nativ nicht unterstützt. In allen anderen bei der Evaluation berücksichtigten Sicherheitsaspekten unterschieden sich die Produkte nicht maßgeblich.

Besonderheiten
Während sich bei GridSphere keine relevanten Alleinstellungsmerkmale feststellen ließen, konnte Liferay mit einer Alfresco Integration und der Unterstützung des WSRP Standards aufwarten. Auch GateIn besitzt neben einer speziellen Sicherheitstechnologie (mittels Token) die angesprochene WSRP Funktionalität. Diese steht zwar derzeit nicht im Fokus der pelican-Anwendung, könnte aber zu einem späteren Zeitpunkt eine Rolle spielen.

Individuelle Portalprofile

Nachdem die Ergebnisse in den einzelnen Kategorien verglichen wurden, soll die Tabelle jetzt spaltenweise analysiert und Produktprofile abgeleitet werden.

GridSphere konnte mit 35 von insgesamt 45 möglichen Punkten eine gute Bewertung erzielen. Stärken der Portallösung liegen in der Unterstützung sämtlicher geforderter Standards und der schlanken Implementierung. Die dreifach gewerteten Kategorien Technische Rahmenbedingungen und Sicherheit wurden in nahezu vollem Umfang erfüllt. GridSphere besaß als einziges Produkt eine Möglichkeit zur Integration in ein Grid, wenn auch nur über eine Zusatzapplikation.

Negativ aufgefallen ist das Fehlen eines offiziellen Forums und das schlichte Standardbenutzungsinterface, das nicht mehr ganz zeitgemäß wirkt. Als einzige Software konnte GridSphere keine Bonuspunkte im Bereich Besonderheiten erlangen. Das liegt darin begründet, dass die Entwickler das Framework bewusst auf das Wesentliche reduzierten, um Entwicklern später möglichst viel Freiraum einzuräumen.

Liferay teilt sich mit ebenfalls 35 von 45 Punkten den ersten Platz mit GridSphere. Funktionell beeindruckt die Lösung vor allem durch die vielen, bei der Installation bereits enthaltenen Portlets und Plugins wie beispielsweise Wikis, Umfragen und eine Google Maps Anbindung. Diese Module können bequem per Drag-and-Drop Funktion

4 dt. „ziehen und loslassen“. Bezeichnet eine Methode, Oberflächenkomponenten mittels gedrückter Maustaste in andere, geeignete Umgebungen zu verschieben.

auf der Oberfläche verankert werden. Die Oberflächenkomponenten sind funktional und ergonomisch gestaltet und entsprechen dem modernen „Web 2.0 Gefühl“. Eine ausführliche Dokumentation ist in großem Umfang und hoher Qualität vorhanden. Der Liferay Nutzer kann auf diverse Wiki-Einträge, Videoanleitungen und Leitfäden zu Themen wie Installation, Administration, Konfiguration und Entwicklung von Portlets zurückgreifen. Besonders hervorzuheben ist ein Tuning Guide, der für jede technische Komponente Einstellungen zur Leistungsoptimierung beschreibt. Ein Nutzerforum und die Möglichkeit, Softwarefehler zu melden und deren Bearbeitung zu verfolgen, runden den positiven Gesamteindruck ab.

Mit diesem überwältigenden Leistungsspektrum, bei dem der überwiegende Teil aller Funktionen bei der pelican-Anwendung jedoch nicht zum Einsatz kommt, gehen mit dem hohen Einarbeitungsaufwand und schlechter Performanz (siehe hierzu 4.1.6) gravierende Nachteile einher.

GateIn erreichte mit 32.5 von 45 Punkten nur einen dritten Platz, wenngleich die Differenz zu den anderen Produkten minimal ist. Ähnlich wie bei Liferay fällt vor allem die moderne und ergonomische Oberflächengestaltung auf. Das Anpassen des Layouts ist sehr komfortabel per Drag-and-Drop Funktion möglich. Bonuspunkte wurden verliehen für die Unterstützung des WSRP-Standards und einer speziellen Token Service API, die Single Sign-On systemweit vereinfacht. Punkte einbüßen musste die Software in den Kategorien Sicherheit (auf Grund mangelnder Loggingfunktionalität und Passworteinstellungen) und Support. Da GateIn noch relativ neu auf dem Markt ist, gibt es außer den ausführlichen User und Reference Guides bisher keine weiteren Leitfäden auf der Herstellerwebseite zu finden.

4.1.6 *Auswahl*

Jedes der Portale besitzt individuelle Stärken und Schwächen. Keiner der Kandidaten war in der Lage, alle formulierten technischen und funktionellen Anforderungen in vollem Maße zu erfüllen. Es sind jedoch auch bei keiner Lösung gravierende Unzulänglichkeiten aufgetreten. Daraus lässt sich schließen, dass alle evaluierten Portale prinzipiell für den Einsatz im pelican-Projekt geeignet wären.

Betrachtet man die Gesamtwertungen der evaluierten Portale, fällt die geringe Varianz der Lösungen auf. GridSphere und Liferay erreichten sogar exakt dieselbe Punktzahl. Ein Grund dafür könnte sein, dass sie sich in der für unser Projekt relevanten Funktionalität wenig unterscheiden und die Unterschiede eher im Detail liegen. Es fällt daher schwer, eine objektive Auswahl zwischen den Produkten nur anhand

des Punktwertes zu treffen. GateIn soll jedoch, als Kandidat mit weniger Punkten als die Konkurrenz, aus Gründen der Vereinfachung nicht in die engere Auswahl einbezogen werden.

Liferay ist funktionell wesentlich umfangreicher als GridSphere. Neben seinem Content Management System (CMS) werden unter anderem ein Workflow Framework und diverse Kommunikationswerkzeuge wie Wikis, Blogs oder Instant Messaging mitgeliefert. Diese Vielfalt wirkte sich bei der Installation allerdings kontraproduktiv aus: Da für die Evaluation nur die formulierten Anforderungen betrachtet werden sollten, erschien der zusätzliche Umfang als unnötiger Ballast, der administrativen Aufwand erforderte.

Wie neben anderen von Koru et al. [2009] gezeigt werden konnte, wächst mit dem Umfang des Quellcodes auch die Fehleranfälligkeit der Software. Das GridSphere Installationspaket umfasste lediglich 18 Megabyte, das entsprechende Liferay Pendant war bereits mit 214 Megabyte groß und ist damit grundsätzlich als fehleranfälliger einzustufen. Als konkretes Beispiel fiel beim Testen ein JavaScript Fehler beim Ausführen im Internet Explorer 7 auf.

Der Größenunterschied wird auch schon an der Zeit deutlich, die zur Inbetriebnahme der Software (engl. deployment) nach jeder Codeänderung, die einen Neustart des Applikationsservers nach sich zieht, benötigt wird. Während das Deployment der GridSphere Applikation etwa 5 Sekunden dauert, braucht die Liferay Software bis zu 100 Sekunden. Das ist ein signifikanter Unterschied, der sich bei den zu erwartenden häufigen Änderungen bei Liferay zu einer hohen Leerlaufzeit für den Programmierer aufsummieren würde. Da GridSphere zudem in Form von GridPortlets über eine vielversprechende Integration in eine Gridumgebung verfügt, fiel die Entscheidung auf die Portalsoftware des GridLab Projekts.

Mit der Entscheidung für GridSphere als Portalframework wurde die Basis für die Implementierung eines pelican-Portlets geschaffen. Der folgende Teil der Diplomarbeit beschäftigt sich mit Vorbereitung, Konzeption und Durchführung der Entwicklung des Portlet-Prototypen. Dabei werden insbesondere applikationsspezifische Schwierigkeiten diskutiert, die bei der Programmierung aufgetreten sind. Die präsentierten Lösungsansätze sollen zukünftige Nach- oder Weiterentwicklungen im Kontext der pelican-Anwendung unterstützen.

4.2.1 *Vorbereitung*

Softwareentwicklung auf Basis existierender Applikationen bedarf stets sorgfältiger Vorüberlegungen. Dazu gehören die Auswahl einer geeigneten Entwicklungsumgebung sowie die Analyse und Integration bereits bestehender Softwarekomponenten. Im konkreten Fall muss außerdem in Form der GridSphere Portalsoftware ein Container bereitgestellt werden, der die entwickelten Komponenten integrieren und verwalten kann.

Entwicklungsumgebung

Eine integrierte Entwicklungsumgebung, oft abgekürzt mit IDE, ist eine Hilfsanwendung zur Programmierung von Software. Sie stellt dem Entwickler verschiedene integrierte Werkzeuge zur Verfügung, welche die Produktivität bei der Erstellung von Quellcode deutlich steigern können. Dazu gehören unter anderem (Quell-)Texteditoren, Compiler, Debugger zur Fehlerfindung und Projektmanagementfunktionen.

Um GridSphere in Eclipse ausführen und anpassen zu können, muss ein neues Projekt der Kategorie `Dynamic Web Project` in der IDE erzeugt werden. Existierende Dateisysteme können über die Importfunktion in das neue Projekt eingebunden werden. Es ist bei der Entwicklung von GridSphere Portlets mit Eclipse darauf zu achten, dass der Erstellvorgang („build“) nicht über die IDE, sondern mittels der im Installationspaket enthaltenen Datei `build.xml` erfolgen muss. Diese sollte so konfiguriert sein, dass alle notwendigen Dateien inklusive Bibliotheken an den richtigen Ort im Unterverzeichnis des Applikationsservers kopiert werden.

Installation von GridSphere

Die Installation der GridSphere Software wurde testweise bereits im Rahmen der Evaluation aus Kapitel 4.1 durchgeführt. Um das Portal auf einem Server installieren zu können, muss eine Java Laufzeitumge-

bung (JRE) der Version 1.5 oder höher vorhanden sein.[5] Der außerdem zum Ausführen des Javacodes auf dem Webserver benötigte Servlet Container Apache Tomcat ist bereits im GridSphere Installationspakets enthalten.[6]

Nach dem Herunterladen muss die enthaltene Ordnerstruktur in den `webapps` Ordner der entsprechenden Tomcat Installation verschoben werden. Zusätzlich muss die Umgebungsvariable `JAVA_HOME` des Betriebssystems auf den Pfad der korrekten Java Version verweisen. Jetzt kann die GridSphere Applikation mit Aufruf der Datei `startup.sh` (beziehungsweise `startup.bat` in einer Windowsumgebung) im Verzeichnis `Tomcat/bin/` gestartet und über einen Internetbrowser aufgerufen werden. Die Standardadresse lautet `http://ip_des_servers:8080/gridsphere/gridsphere`.

Im folgenden Dialog kann ausgewählt werden, ob GridSphere eine interne Datenbank für die Datenhaltung benutzen soll. Wird die Nutzung einer eigenen Datenbank präferiert, wie mySQL im Fall von pelican, müssen entsprechende Verbindungsparameter angegeben werden. Der letzte Schritt in der Konfiguration der GridSphere Anwendung ist die Erstellung eines Administratorprofils.

Analyse des bisherigen Prototypencodes

Auf Basis der Programmiersprache Java und damit verknüpfter Webtechnologien (JavaServer Faces (JSF), JavaScript) wurde innerhalb des pelican-Projekts ein Webservices-Prototyp entwickelt. Dadurch konnten erste Erfahrungen mit der caBig Infrastruktur gesammelt und den Mitarbeitern des TRR in Form eines anschaulichen Beispiels präsentiert werden.

Der festgelegte Ablauf des Programms führt den Benutzer durch eine Reihe von Datenbankabfragen, an deren Ende eine tabellarische Anzeige der kombinierten Ergebnisse stehen. Für jede der vier Abfragen (in den TRR-Projekten entstandene ACGH-Daten, RNA-Daten, Methylierungsdaten sowie Informationen über das Probenkollektiv) können spezifische Einschränkungen vorgenommen werden, beispielsweise eine maximale Sequenzlänge im ACGH-Datensatz.

Der Prototyp soll im Rahmen dieser Diplomarbeit in eine Portletapplikation umgeschrieben werden und in das pelican-Portal eingefügt werden. Dabei sollen funktionelle Änderungswünsche der TRR-Mitarbeiter an das Programm berücksichtigt werden. Als Erfolgskriterium für den Einsatz des Prototypen als Portlet soll eine erfolgreiche Kommunikation mit der serverseitigen Datenbank in

5 verfügbar zum Beispiel unter `http://www.java.com/de/download/`
6 verfügbar unter `http://www.gridsphere.org/gridsphere/gridsphere/`

Form von Datenlieferungen und deren adäquate Anzeige dienen.

Der Quellcode beinhaltet:

- 12 eigene Java-Klassen (eigentliche Programmlogik)
- 17 Hilfsklassen
- 12 JavaServer Pages (JSP) Dateien
- 1 Cascading Style Sheet (CSS) zur Präsentation
- diverse Konfigurationsdateien (beispielsweise für JSF oder Logging-Funktionen)
- eine Vielzahl an externen caBig Bibliotheken

Der Quellcode kann mittels geeigneter Subversion Software (etwa das Subclipse Plug-in für Eclipse) und entsprechender Berechtigung vom Projektserver [7] bezogen werden.

4.2.2 *Entwurf der Portletapplikation*

Bevor die Anwendung technisch umgesetzt werden kann, muss der Ablauf des bisherigen Prototypen an die neue Architektur und Änderungswünsche der TRR-Mitarbeiter angepasst werden.

Anpassung des Workflows

Verbesserungspotenzial sehen die Forscher vor allem in der mangelnden Flexibilität bei den einzelnen Analyseschritten, die derzeit komplett durchlaufen werden müssen. Die Reihenfolge der Datenbankabfrage sei zudem aus medizinischer Sicht weniger sinnvoll. Es fehlt bisher eine Suche nach Genen beziehungsweise Genloki (von lat. locus, der Ort) innerhalb der Informationsplattform. Diese Art von Abfrage sei aber die weitaus häufigste in der täglichen Arbeit der Projektmitarbeiter.

Der statische Programmablauf soll nun mit Hilfe von Portlets flexibel gestaltet werden. Das Portletmodell erlaubt eine genaue Selektion der benötigten Funktionen. Einzelne, nicht benötigte Datendienste können vom Benutzer ausgeblendet werden. Es bietet sich deswegen an, die vier geschilderten Abfragen in eigenen Diensten zu kapseln und als Portlets zu präsentieren. Außerdem soll ein dediziertes Portlet eine Auswahl derjeniger Datensätze ermöglichen, die miteinander korreliert werden sollen.

7 `https://host-trr77-imbi.urz.uni-heidelberg.de/repos/pelican`

Als Korrelation bezeichnet man einen statistischen Zusammenhang: Sie beschreibt, „[...] wie eng zwei Faktoren miteinander verknüpft sind und sich gemeinsam verändern, bzw. wie gut der eine Faktor das Auftreten des anderen vorhersagt [Myers 2008, S.30].“ Durch solch eine Verknüpfung können die Forscher des TRR unter Umständen neue Erkenntnisse über die molekularen Zusammenhänge gewinnen.

Konzeption der Benutzeroberfläche

Für die Umsetzung des vorgeschlagenen Workflows inklusive der zusätzlichen Suchfunktion nach Genen werden sechs Portlets benötigt:

- Search Portlet
- ACGH Portlet
- RNA Portlet
- Methylation Portlet
- Collective Portlet
- Correlation Portlet

Nach dem erfolgreichen Login sieht der Nutzer zunächst alle sechs Portlets, sofern er keine persönlichen Anpassungen vorgenommen hat. Im *Search Portlet* befinden sich Eingabefelder für die Suche nach Gensymbolen, einzelnen Chromosomen, einem Lokusbereich sowie der Strangrichtung. Per Mausklick auf die entsprechende Schaltfläche werden diese Parameter an die anderen Portlets übermittelt.

ACGH Portlet, *RNA Portlet* und *Methylation Portlet* sind zunächst leer bis auf eine Suchmaske für ihre spezifischen Einschränkungen der Parameter. Diese ermöglichen eine Filterung nach maximaler Sequenzlänge für das ACGH Portlet sowie Minimal- und Maximalwerte für die RNA- und Methylierungsergebnisse. Sobald per Schaltfläche eine Suchanfrage im System gestartet wurde, erscheint die jeweilige Ergebnisanzeige kurze Zeit später in tabellarischer Form in den entsprechenden Portlets. Wurde zuvor im Search Portlet nach einem Gen oder DNA-Bereich gefiltert, so wird auch diese Einschränkung bei der Datenbankabfrage berücksichtigt und nur Ergebnisse für das gesuchte Gen oder den gesuchten DNA-Bereich angezeigt.

Im *Collective Portlet* lässt sich die Gesamtheit der Proben anzeigen und nach klinischen Parametern filtern (unter anderem die eindeutige Identifikationsnummer, Geschlecht, Materialart, Alter und Größe der Probe). Für jeden Parameter wird dabei eine geeignete Darstellung gewählt, beispielsweise ein Balkendiagramm für die Altersverteilung.

Das *Correlation Portlet* beeinhaltet eine Anzeige aller Datensätze, die im bisherigen Sitzungsverlauf aus der Datenbank abgerufen wurden. Hat der Benutzer bereits einen RNA- und einen ACGH-Datensatz abgefragt, erscheinen im Korrelationsportlet automatisch zwei Auswahlkästen (engl. checkboxes) mit den jeweiligen Bezeichnungen. Diese können nun separat angewählt werden, um den entsprechenden Datensatz in die Gesamtkorrelation einzubeziehen.

Abbildung 4.6 stellt anschaulich dar, wie eine Auswahl der genannten Portlets innerhalb der GridSphere Umgebung angeordnet werden könnte. In diesem Szenario hat der Nutzer die Portlets für Suche, ACGH-Daten, RNA-Daten und Korrelation aktiviert. Im Suche Portlet wurden diverse Filterparameter (beispielsweise Chromosom 21) eingegeben, die von den anderen Portlets registriert und in die Datenbankabfrage einbezogen werden (siehe ACGH-Datensatz für das entsprechende Chromosom). Nach erfolgreicher Lieferung können die ACGH Daten im Korrelationsportlet für eine datensatzübergreifende Analyse ausgewählt werden.

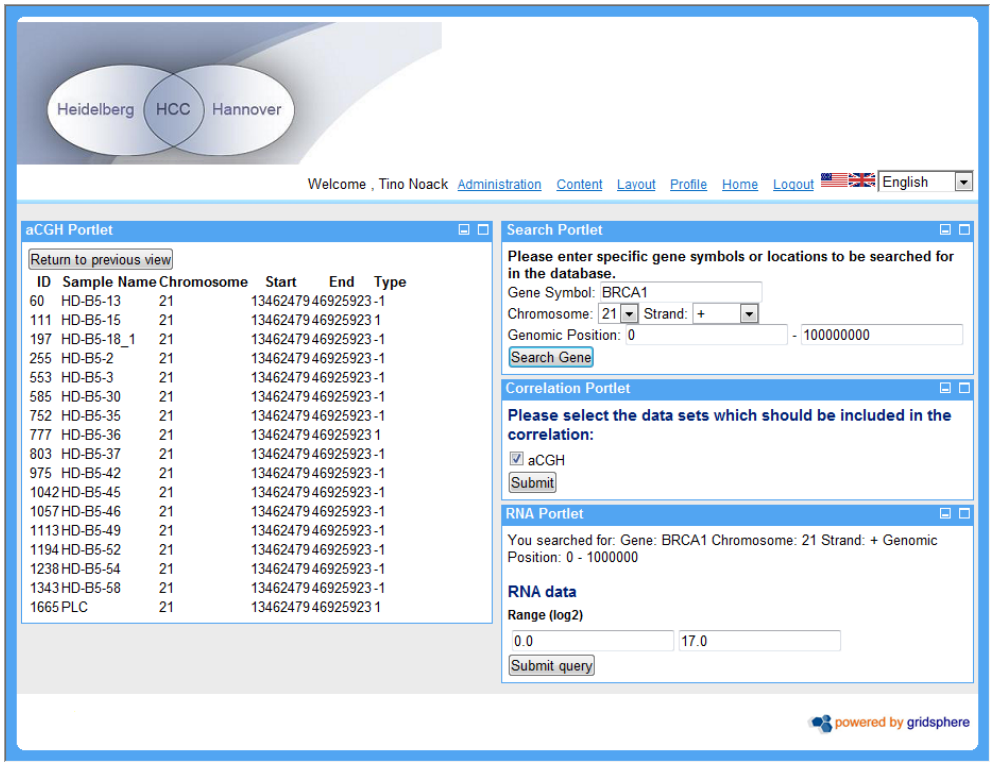

Abbildung 4.6: Portlet Konzept in Portalumgebung.

Technische Umsetzung

Es ist prinzipiell möglich, die bestehende Applikation als Portletanwendung im GridSphere Portal laufen zu lassen. Dafür müssen folgende Schritte ausgeführt werden:

Anpassung der Konfigurationsdateien. Zwei dieser so genannten Deployment Deskriptoren im XML-Format müssen nach JSR-168 im WEB-INF Ordner der Portletapplikation existieren. In der portletspezifischen Datei `portlet.xml` werden alle vier vorgestellten Portlets der Anwendung und deren Eigenschaften und Ressourcen definiert. Die Datei `web.xml` enthält Informationen zur Bereitstellung der Anwendung im Tomcat, unter anderem benötigte Ressourcen, Referenzen und Parameter. Hier muss ein spezieller Eintrag vorgenommen werden (siehe Listing 4.1), der die pelican-Applikation mit der GridSphere Hauptanwendung verknüpft. Der Quellcode dieser Datei befindet sich in Anhang C. Für detaillierte Informationen zum Thema Deployment Deskriptoren in Portalanwendungen sei an einschlägige Literatur verwiesen [Zörner 2006, S.32f].

Listing 4.1: GridSphere Referenz in web.xml.

```
<servlet>
        <servlet-name>PortletServlet</servlet-name>
        <servlet-class>org.gridsphere.provider.portlet.jsr.
            PortletServlet</servlet-class>
</servlet>
```

Entfernung überflüssiger HTML-Tags. Eine Portalapplikation zeichnet sich dadurch aus, dass die angezeigte HTML Seite dynamisch zur Laufzeit erzeugt wird. Portlets werden lediglich als Codefragmente eingefügt. Sämtliche benötigte Komponenten einer HTML-Seite, wie Dokumenttypdeklaration und `<head>` beziehungsweise `<body>` Tags werden von der GridSphere Portalsoftware generiert. Der pelican-Prototyp besteht jedoch aus einzelnen HTML Seiten, die beim Programmablauf nacheinander aufgerufen werden. Damit das Portal standardkonforme HTML Seiten ohne doppelte Definitionen und Tags erzeugt, müssen diese aus dem pelican-Quellcode entfernt werden.

Import von JavaScript oder CSS. Sollte die Notwendigkeit bestehen, JavaScript oder CSS am Anfang einer Seite einzubinden, muss ebenfalls auf das beschriebene Verhalten eines Webportals Rücksicht genommen werden. Zusätzliche derartige Dateien müssen über einen speziellen Funktionsaufruf in der Portletklasse eingebunden werden (siehe Listing 4.2). Alternativ können die Ressourcen in eine spezielle

GridSphere Klasse integriert werden[8]. Dies wurde für das vorliegende Projekt nötig, da der zuvor entwickelte Prototyp eine Vielzahl von JavaScript Elementen enthält.

Listing 4.2: Methoden zum Import von CSS und JavaScript.

```
% Aufrufen, um externe CSS-Dateien einzubinden.
renderResponse.setProperty("CSS_HREF", request.getContextPath() +
"gridsphere/css/style.css");

% Aufrufen, um externe JavaScript Bibliotheken einzubinden.
renderResponse.setProperty("JAVASCRIPT_SRC", request.
    getContextPath() +
"gridsphere/js/bibliothek.js");

% Aufrufen, um JavaScript Methoden als "onload" Attribut zum
% HTML <body> hinzuzufügen.
renderResponse.setProperty("BODY_ONLOAD", "funktion()");
```

JavaServer Faces. Die Verwendung von JSF in Kombination mit Portlet-Technologie ist nicht unproblematisch. Die Komponenten beider Technologien besitzen unterschiedliche Lebenszyklen, die aufeinander abgebildet werden müssen. Dieses „Mapping“ wird von einer so genannten JSF Portlet Bridge übernommen. Einen Überblick über die Architektur der Bridge liefert Bosch [2009]. Für die pelican-Applikation wird als Implementierung der Bridge die MyFaces Tomahawk Bibliothek der Apache Software Foundation benutzt[9]. Es ist darauf zu achten, alle notwendigen Bibliotheken in das Projekt einzubinden.

Der erhöhte Programmierungsaufwand lohnt sich dennoch, denn die JavaServer Faces Technologie vereinfacht die Navigation zwischen den JSP Seiten und verwaltet die Datenobjekte (JavaBeans) eigenständig. Damit die JSF Komponenten des pelican-Prototyps auch im Portalumfeld funktionieren, müssen erneut die Deployment Deskriptoren (`web.xml`, `portlet.xml`) angepasst werden. Darüber hinaus muss die JSF spezifische Datei `faces-config.xml` alle von den Portlets verwendeten Datenobjekte und Navigationsregeln definieren.

Reihenfolge des Ladens. Es ist notwendig, für jede Portletapplikation eine leere Datei gleichen Namens anzulegen. Diese wird von GridSphere zum Laden der Module in der korrekten Ladereihenfolge benötigt. Die Dateiendung (als Zahl) gibt dabei die Priorität an. Das GridSphere Grundmodul heißt stets `gridsphere.1` und wird beim Start der Anwendung zuerst geladen. Danach folgen die anderen Ap-

8 `gridsphere.src.org.gridsphere.layout.view.brush.Page.java`
9 `http://myfaces.apache.org/tomahawk/index.html`

plikationen in aufsteigender Reihenfolge entsprechend, beispielsweise `PELICAN.2`. Unter Linux als Betriebssystem muss sich diese Datei im Ordner `$HOME/.gridsphere/portlets/` befinden.

Die Klassenstruktur

Herzstück des Klassenkonzepts ist die Java Klasse `AllData.java`. Für jede Sitzung (engl. session) eines Benutzers wird ein Objekt dieser Klasse erzeugt. Es enthält eine Sammlung (engl. collection) aller Datenobjekte, die während dieser Sitzung durch Datenbankabfragen erzeugt wurden. Jede dieser Abfragen stellt ein Objekt der Klasse `DataContainer.java` dar. DataContainer können beliebige Typen von Datensammlungen sein, wie beispielsweise RNA-Datensätze. Außerdem werden in AllData Objekten alle Filterparameter als HashMap verwaltet, die bei einer Korrelation der Datensätze Anwendung finden sollen. Für die formulierten Anforderungen an die prototypische Applikation sind die Parameter Gensymbol, diverse Positionsangaben und eine Probenidentifikation notwendig.

Jeder der vier Abfragetypen besitzt eine eigene Datenklasse. In ihnen ist die jeweilige Programmlogik enthalten, beispielsweise die Generierung der Datenbankabfragen als Kommandos der caGrid Query Language (CQL).

Objekte der Klasse `Correlation.java` speichern die Auswahl des Nutzers im Korrelationsportlet in Arrays. Diese Listen können über get()-Methoden vom AllData Objekt ausgelesen und entsprechend verarbeitet werden. Im Search Portlet verwendete Suchparameter werden in der Klasse `Gene.java` verwaltet.

Abbildung 4.7 visualisiert das Zusammenspiel der Java Klassen schematisch. Es sind jedoch nur die wichtigsten Parameter und Operationen aufgeführt. AllData stellt eine Komposition beliebig vieler DataContainer dar. Diese DataContainer können Objekte der verschiedenen Datenklassen beinhalten. Objekte der Korrelationsklasse sind direkt mit erzeugten Datencontainern verknüpft. Die Genklasse stellt Filterparameter zur Verwaltung in AllData bereit.

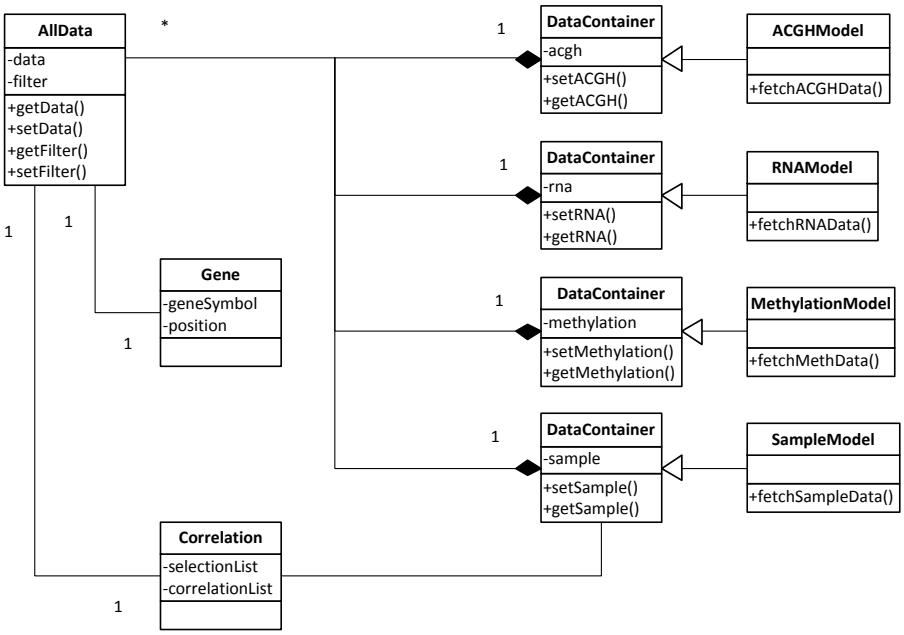

Abbildung 4.7: Klassendiagramm der Portletapplikation.

Der gesamte Quellcode für die pelican-Anwendung, der sowohl die Javaklassen als auch JSP- und Konfigurationsdateien beinhaltet, befindet sich in einem eigenen Subversion Verzeichnis des Projektservers[10].

4.2.3 *Implementierungsdetails*

Der folgende Abschnitt beschreibt weitere Arbeitsschritte, die für die Integration des Prototypen in GridSphere vorgenommen wurden. Insbesondere werden aufgetretene Schwierigkeiten dokumentiert und Lösungsansätze diskutiert.

Inter-Portlet-Kommunikation

Eine große Herausforderung bei dieser Anwendung besteht in der Übermittlung von Informationen zwischen den Portlets, auch als Inter-Portlet-Kommunikation bezeichnet. Diese Informationen können beispielsweise eingegebene Parameter oder ganze Datenobjekte sein. Doch wie sollen die, im Gegensatz zum Prototypen, voneinander unabhängigen Funktionsbausteine miteinander kommunizieren?

Es hat sich herausgestellt, dass die von GridSphere implizierte Unterstützung der JSR-268 Spezifikation zum jetzigen Zeitpunkt lediglich

10 `https://host-trr77-imbi.urz.uni-heidelberg.de/repos/pelican/PELICAN`

in Entwicklung ist. In dieser zweiten Spezifikation werden jedoch explizit Möglichkeiten zur Inter-Portlet-Kommunikation definiert. Da GridSphere keine alternative proprietäre Lösung anbietet, muss in der pelican-Anwendung folgende Hilfskonstruktion benutzt werden:

Wird der Sichtbarkeitsbereich (engl. scope) der Datenobjekte in der Datei faces-config.xml entsprechend definiert, können deren Parameter portletübergreifend gesetzt und ausgelesen werden. Es werden dazu so genannte *value binding* Objekte benutzt. Diese dienen der Verknüpfung von sitzungsspezifischen Objekten. In folgendem Quelltextauszug wird das AllData Objekt der aktuellen Sitzung („facesContext“) mit dem Objekt einer Datenmodellklasse verknüpft.

Listing 4.3: Inter-Portlet-Kommunikation durch Value Binding.

```
FacesContext facesContext = FacesContext.getCurrentInstance();
Application application = facesContext.getApplication();

ValueBinding binding = application.createValueBinding("#{allData
    }");
AllData allData = (AllData)binding.getValue(facesContext);
```

Diese Methode wird in der pelican-Anwendung eingesetzt, um Parameter der Suche an die Datenportlets zu übertragen und im Korrelationsportlet auf eine Liste bereits abgerufener Datensätze zurückzugreifen. Allerdings ist diese Art der Informationsübertragung nur applikationsweit möglich. So lange also keine in JSR-268 definierte Methode von GridSphere unterstützt wird, müssen sich alle Portlets innerhalb einer Applikation befinden. Das hat Auswirkungen auf die Standardkonformität der entwickelten Portlets. Sie können vorerst nicht ohne Weiteres unabhängig, zum Beispiel in anderen Portalen, mit vollem Funktionsumfang laufen.

Darstellung mehrerer Portlets

Beim Test der Anwendung kam es zu unerwünschtem Verhalten bei der parallelen Nutzung mehrerer Portlets. Lediglich die Nutzeraktionen im ersten Portlet der Seite wurden ordnungsgemäß ausgeführt. Die Schaltflächen der anderen Portlets reagierten gar nicht oder mit Fehlern in der Anzeige. Es fiel auf, dass sich die verschiedenen Portlets fälschlicherweise eine interne Komponentenidentifikation (`View_ID`) teilten. Dieser Fehler ist laut entsprechenden Einträgen in entsprechenden Fehlerverwaltungsseiten[11] mit hoher Wahrscheinlichkeit auf veraltete Versionen von JSF (1.1) und der Bridge-Implementierung

11 vgl. `https://issues.apache.org/jira/browse/MYFACES-650`, zuletzt abgerufen am 27.04.2011

(MyFaces 1.1.9) zurückzuführen.

Es hat sich herausgestellt, dass die Nutzung von Komponenten aus der caGrid Infrastruktur einige Einschränkungen mit sich bringt. Die darin bereitgestellten Werkzeuge sind optimiert für den Betrieb auf einem Tomcat Server der Version 5.5. Diese Version unterstützt aber nicht die aktuellsten Versionen der JSF Technologie (2.0) und Bridge-Implementierung (MyFaces 3.0). Der Versuch, diverse andere Versionen der Technologien in die Anwendung einzubinden, war bislang nicht erfolgreich.

Damit dennoch die Funktionalität der Portlets entwickelt und das Konzept verifiziert werden konnte, wurden übergangsweise sämtliche Funktionen in ein gemeinsames Portlet integriert (siehe Abbildung 4.8). Über eine Leiste von Schaltflächen am oberen Rand kann der Nutzer von jeder Ansicht zu allen Funktionen navigieren. Die Flexibilität im Prozessablauf bleibt gewahrt. Dazu mussten lediglich die Navigationsregeln in der Datei `faces-config.xml` angepasst und statt sechs Portlets nur eines in der `portlet.xml` definiert werden.

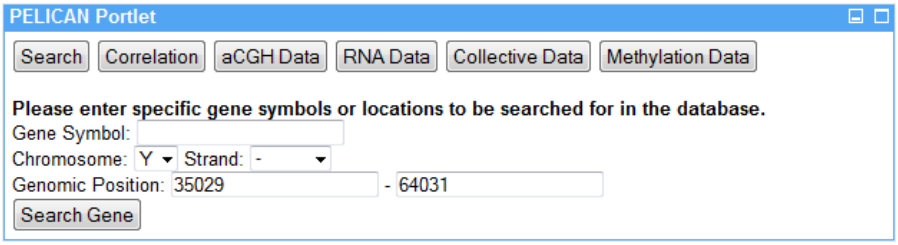

Abbildung 4.8: Kombination der Funktionen in einem Portlet.

Mit Aktualisierung der caGrid Infrastruktur kann in absehbarer Zukunft auch die verwendete JSF Version angepasst und der Darstellungsfehler behoben werden. Durch Wiederherstellung der ursprünglichen Konfigurationsdateien, die im Quellcode nach wie vor enthalten sind, können die Portlets mit wenig Aufwand wieder separiert werden.

Layout des Portals anpassen

Es gibt drei Möglichkeiten, das Erscheinungsbild des GridSphere Portals anzupassen. Die erste ist das Layoutmanagement direkt in der Anwendung. Hier können vom Administrator neben den Portlets auch Komponenten wie Banner am Beginn oder Ende der Portalseite in die Ansicht eines einzelnen Nutzers eingebunden werden.

Die zweite Möglichkeit ist die Erstellung von eigenen so genannten Themen (engl. themes) auf Basis der CSS-Technologie. Die Definition

der Themen müssen im Verzeichnis `Tomcat/webapps/gridsphere/themes/` hinterlegt werden. Als Vorlage kann das bereits vorhandene „default“ Thema kopiert und die relevanten Merkmale abgeändert werden.

Die letzte Möglichkeit ist die Konfiguration von rollenspezifischen Layout Deskriptoren im xml-Format, zum Beispiel des Standard Nutzerprofils. Die Deskriptoren befinden sich im Verzeichnis `Tomcat/webapps/gridsphere/WEB-INF/CustomPortal/layouts`. Für die pelican-Informationsplattform wurde das offizielle Logo des TRR als Seitenbanner eingebunden.

Inkompatible caBig Bibliotheken

Es hat sich als problematisch herausgestellt, stets jede der 129 Klassenbibliotheken der caBig Version 1.3 in eigene Applikationen einzubinden. Abgesehen von Performanzeinbußen durch das Laden unnötiger Bibliotheken traten häufig Fehler auf, die durch eine inkorrekte Ladereihenfolge der Klassen hervorgerufen wurden. Darauf hat man bei der Verwendung mitgelieferter GridSphere Installationsdateien nur indirekten Einfluss.

Um diese Probleme in Zukunft zu umgehen, wurden in einem iterativen Verfahren alle zum Ausführen des Prototypen unbedingt notwendigen caBig-Bibliotheken identifiziert, die im Anhang A zur zukünftigen Einsichtnahme aufgelistet sind. Zum Vergleich: Statt der ursprünglichen 129 reichen lediglich 24 Bibliotheken für die erfolgreiche Installation der pelican-Anwendung.

Logging

Sowohl GridSphere als auch der pelican-Prototyp nutzen, ursprünglich unabhängig voneinander, das log4j Framework der Apache Software Foundation. Es stellt Klassen zur Verfügung, die Ausgaben der Applikation zur Laufzeit in Protokollen festhalten. Inhalt, Umfang, Format und Ort können flexibel vom Entwickler angepasst werden. Damit wird die Suche nach Fehlern (so genanntes Debugging) wesentlich vereinfacht und das Verhalten der Software transparenter gestaltet.

Damit Log-Ereignisse des pelican-Portlets von GridSphere protokolliert werden, muss folgende Zeile in die Datei `log4j.properties` eingefügt werden:

```
log4j.logger.de.trr77=DEBUG
```

Der Parameter `DEBUG` definiert das aktuelle Loglevel und kann entsprechend den Bedürfnissen des Entwicklers angepasst werden. Beim Loglevel `ERROR` werden beispielsweise nur Fehler im Programmablauf

protokolliert. Für weiterführende Informationen zum Thema log4j sei an die offizielle Dokumentation zum Thema[12] verwiesen.

Benutzung der Browserchronik

Die Benutzung der „Zurück"-Schaltfläche im Browser ist mit Risiken verbunden. Angenommen, der Benutzer verändert durch eine Abfrage ein Datenobjekt auf Serverseite und navigiert danach über die „Zurück"-Schaltfläche auf die vorhergehende Ansicht. Der Server bekommt diese Aktion aber nicht übermittelt, denn der Browser ruft lediglich eine gespeicherte Seite aus dem Cache (browserinterner Speicher) auf. Der Zustand der Datenobjekte auf Server- und Clientseite ist zu diesem Zeitpunkt unter Umständen nicht mehr konsistent.

Dieses Verhalten führte bereits in der Vorgängerapplikation zu Problemen. Es stellt eine generelle Herausforderung der Kommunikation zwischen Client und Server dar. In der pelican-Anwendung konnte zunächst nur eine temporäre Lösung durch eigene Schaltflächen zur rückwärtigen Navigation für jedes Portlet bereitgestellt werden. Die Nutzer müssen vorerst auf die Problematik hingewiesen werden und die Nutzung der Browserchronik innerhalb der Applikation nach Möglichkeit vermeiden. Langfristig könnte das Problem behoben werden, indem der korrekte Zustand auf Serverseite aus eindeutigen Informationen, die in Links oder JSP-Actions hinterlegt sind, rekonstruiert wird.

Begrenzung des Datenstroms

Beim Testen der Datenbankabfragen tauchten Fehlermeldungen auf, sobald der Ergebnisdatensatz mehr als 1000 Einträge aufwies. Laut caGrid Hilfeseite[13] wird der Datensatz jedoch vollständig übertragen, sobald ein Zeiger (engl. iterator) nacheinander alle Einträge des Datenobjekts durchläuft. Praktisch bedeutet diese Einschränkung aber, dass `count`-Abfragen, die nur die Ergebnismenge zurückliefern sollen, bei mehr als 1000 Einträgen nicht funktionieren. Es handelt sich hierbei um eine Einstellung auf Serverseite, die den Datenstrom bei einer Abfrage begrenzt. Eine mögliche Lösung besteht in der Anpassung des Parameters `resultCountPerQuery`, welcher sich in der Datei `application-config.xml` der caGrid Applikation befindet und standardmäßig auf den Wert 1000 eingestellt ist. Eine Überprüfung der Lösung konnte aus Zeitgründen im Rahmen der Arbeit nicht durchgeführt werden.

12 `http://logging.apache.org/log4j/1.2/manual.html`, zuletzt abgerufen am 05.01.2011

13 http://wiki.cagrid.org/display/support/Troubleshooting, zuletzt abgerufen am 25.04.2011

5 ZUSAMMENFASSUNG UND DISKUSSION

Im folgenden Kapitel werden die Ergebnisse der vorliegenden Arbeit zusammengefasst und im Hinblick auf Erreichung der formulierten Ziele und Vorgehen kritisch betrachtet.

5.1 BEANTWORTUNG DER FRAGESTELLUNG

Zusammenfassend können nun die Fragen aus Kapitel 1 wie folgt beantwortet werden.

1.1 Welche Methodik eignet sich für eine systematische Evaluation von Portalsoftware?

Erste Aufgabe der vorliegenden Arbeit war es, eine Portalsoftware auszuwählen, die das geplante flexible Benutzerschnittstellenkonzept unterstützen kann. Dazu musste zunächst eine Methodik entwickelt werden, um potenzielle Lösungen miteinander vergleichen und eine möglichst objektive Entscheidung fällen zu können. Wie eine Literaturrecherche zum Thema Evaluation von (Portal-)Software in Kapitel 4.1.1 zeigte, ist der Einsatz von Checklisten diesbezüglich verbreitet und zweckmäßig. In dieser systematisch strukturierten Liste konnten ermittelte Anforderungen kategorisiert und gewichtet werden. Über ein Punktesystem ließen sie sich zueinander in Beziehung setzen.

Die Erfüllung der Kriterien wurde mittels einer Testinstallationen und anhand verfügbarer Dokumentation für jede evaluierte Software überprüft und in drei Graden bewertet.

1.2 Welchen funktionellen und technischen Anforderungen unterliegt die Benutzerschnittstelle?

Kapitel 4.1.3 beschreibt die fünf genutzten Kategorien und deren Inhalte im Detail. Die Einteilung orientiert sich an vorhergehenden Studien, berücksichtigt aber stets die spezifische Relevanz für die pelican-Anwendung. Insbesondere finden oft formulierte Anforderungen wie Dokumentenaustausch aus verschiedenen Gründen keine Berücksichtigung, dafür wurden mit *Support* und *Besonderheiten* weitere Kategorien eingeführt.

Die Anforderungen unterteilen sich in solche technischer und solche funktioneller Natur. Erstere werden in der Kategorie *Technische*

Rahmenbedingungen aufgeführt und gewährleisten beispielsweise die Kompatibilität mit der IT-Infrastruktur des Projekts. Die funktionellen Anforderungen aus den Kategorien *Support*, *Layoutmanagement*, *Sicherheit* und *Besonderheiten* wurden formuliert, um das angestrebte flexible Oberflächenkonzept, erweitert um relevante Sicherheitsaspekte, umsetzen zu können. Insgesamt wurden siebzehn Kriterien aufgestellt, die in die Bewertung eingeflossen sind.

1.3 Welche Portalsoftware-Lösungen sollen in die Evaluation einbezogen werden?

Um die Evaluation in angemessener Zeit und Qualität durchführen zu können, beschränkt sie sich auf die drei Open Source Portallösungen GridSphere, Liferay und GateIn Portal. In Kapitel 4.1.4 werden die Produkte im Einzelnen vorgestellt und Argumente für ihre Berücksichtigung im Auswahlverfahren aufgeführt.

1.4 Erstellung einer Bewertungsmatrix unter Berücksichtigung der Kriterien aus 1.2

Es wurde eine Matrix erstellt, in der die Bewertung der drei Softwarelösungen in tabellarischer Form dokumentiert ist. In der ersten Spalte sind die in gewichtete Kategorien unterteilten Evaluationskriterien aufgeführt. In den nachfolgenden Spalten werden den jeweiligen Kandidaten für den Grad der Erfüllung der Kriterien Punkte vergeben, diese mit dem kategoriespezifischen Gewicht multipliziert und pro Spalte aufsummiert.

1.5 Welche Softwarelösung ist für den Einsatz in der Integrationsplattform am ehesten geeignet?

Die Auswertung der Kriterienmatrix allein nach Punkten erlaubte noch keine Entscheidung für eine Portalsoftware, da zwei von ihnen die gleiche Gesamtpunktzahl erreichen konnten. Es wurden daher weitere praktische Erwägungen wie die zum Start der Anwendung benötigte Zeit und unnötiger Konfigurationsaufwand hinzugezogen. Aus diesen Überlegungen heraus wurde GridSphere als zu verwendende Portalsoftware gewählt.

2.1 Erstellung eines flexiblen Benutzerschnittstellenkonzepts.

In Kapitel 4.2.2 wird ein Konzept einer Benutzerschnittstelle für die pelican-Anwendung vorgestellt. Darin werden verschiedenartige Dienste (Datendienste, Analysedienste, Darstellungsdienste) gekapselt und unabhängig voneinander dargestellt. Der Benutzer kann seinen

Bedürfnissen entsprechend auf Darstellung und Anordnung der Module Einfluss nehmen. Es wurde ein Klassendiagramm entworfen, das die Programmlogik der Dienste als eine Menge von Objekten mit Eigenschaften und Funktionen sowie deren Beziehung zueinander darstellt.

2.2 Lässt sich das Konzept unter Verwendung der ausgewählten Portalsoftware prototypisch umsetzen?

GridSphere bietet die Möglichkeit, gekapselte Funktionen als Portlets in eine homogene Oberfläche zu integrieren. Das Layout der Anwendung kann dem pelican-Design angepasst werden. Durch die Erstellung verschiedener Nutzerprofile lässt sich eine Personalisierung von Präsentation und Funktionen verwirklichen.

2.3 Lassen sich bereits implementierte pelican-Funktionen als Module in die Oberfläche integrieren?

Kapitel 4.2.2 stellt einen Leitfaden zur Verfügung, der die Umwandlung einer eigenständigen Applikation wie dem pelican-Prototypen in voneinander unabhängige Module beschreibt. Es konnte in der Praxis gezeigt werden, dass sich die Dienste als Portlets in eine Portalumgebung integrieren lassen und mit allen damit verbundenen Vorteilen zur Verfügung stehen. Durch einen Versionskonflikt zwischen verwendeten Technologien musste auf eine Zwischenlösung zurückgegriffen werden, bei der die Funktionalität der Anwendung in einem Portlet zusammengeführt wurde. Die Flexibilität der Applikation bleibt jedoch erhalten und der modulare Aufbau lässt sich ohne Weiteres wieder herstellen, sobald der Versionskoflikt behoben wird.

2.4 Welche applikationsspezifischen Besonderheiten traten auf?

Besondere Herausforderungen bei der Implementierung der Anwendung waren eine funktionierende Inter-Portlet-Kommunikation, die konfliktfreie Einbindung der caBIG-Bibliotheken und spezifische Fallstricke der GridSphere Software. In Kapitel 4.2.3 wurden diese Probleme identifiziert und praktische Lösungsansätze geliefert.

5.2 ERFÜLLUNG DER FORMULIERTEN ZIELE

Im Folgenden wird das Erreichen der Ziele der Arbeit untersucht und hinsichtlich der Vorgehensweise kritisch betrachtet.

Ziel 1: Evaluation und Auswahl einer Portalsoftware für die Verwendung innerhalb der pelican-Applikation.

Die Portalsoftware GridSphere wurde nach sorgfältiger Evaluation dreier potenzieller Lösungen für am besten geeignet erklärt, als Rahmenkomponente der pelican-Benutzerschnittstelle eingesetzt zu werden. Im Bereich der Open Source Portallösungen findet sich noch eine Vielzahl weiterer Anbieter, die jedoch aus Zeitgründen nicht berücksichtigt werden konnte.

Das in der Arbeit verwendete Verfahren einer Checkliste beziehungsweise Kriterienmatrix ist im Bereich der Softwareevaluation geeignet [Stufflebeam 2001]. Die Auswahl der Evaluationskriterien ist Ergebnis einer Anforderungsanalyse, die spezifisch für die IT-Infrastruktur der TRR-Integrationsplattform durchgeführt wurde. Diesbezüglich bereits publizierte Checklisten wie von Dion Goh et al. [2008] konnten dabei lediglich als Orientierung dienen und mussten zum Großteil angepasst werden. Die entwickelte Checkliste besitzt deswegen keine Allgemeingültigkeit, kann jedoch in modifizierter Form für eventuelle spätere Evaluationsvorhaben Verwendung finden.

In einigen Fällen war eine persönliche Überprüfung des formulierten Kriteriums nicht in angemessener Zeit möglich. Insbesondere die Protokollfunktionalität bedurfte häufig eines erhöhten Konfigurationsaufwands. In diesen Fällen wurden entsprechende Informationen der jeweiligen Produktdokumentation entnommen. Es ist jedoch vorstellbar, dass relevante Informationen nicht ausreichend dokumentiert waren oder trotz sorgfältiger Prüfung nicht gefunden wurden. Beispielsweise konnte die Internetseite der Liferay Programmierschnittstelle zum Zeitpunkt der Evaluation wiederholt nicht aufgerufen werden. Dies führte zu Punktabzug, auch wenn die Funktion vermutlich tatsächlich vorhanden war.

Im Endergebnis wiesen die ermittelten Gesamtwertungen der Produkte eine geringe Varianz auf: Zwei Kandidaten erreichten 35 Punkte, der Drittplatzierte noch 32.5. Als Folge des uneindeutigen Ergebnisses mussten weitere Kriterien herangezogen werden. In Zukunft könnte dies durch eine weitere Differenzierung der Kriterien oder Einführung neuer Kategorien von vornherein vermieden werden. Das Ergebnis verdeutlicht aber auch, dass sich die gängigen Portallösungen in der

Funktionalität gleichen und die spezifizierten Anforderungen in hohem Maß erfüllen.

Ziel 2: Entwicklung eines flexiblen Benutzerschnittstellenkonzepts für die Integrationsplattform.

Es wurde ein Konzept erstellt, wie sich durch die Nutzung von Portlettechnologie beliebige Funktionen kapseln und in einer homogenen Benutzeroberfläche anzeigen lassen. Die Oberfläche besitzt ein einheitliches, mit dem pelican-Logo und entsprechendem Schriftzug versehenes Rahmenlayout, kann jedoch auf Nutzerebene bei Bedarf beliebig angepasst werden. Die Seitennavigation innerhalb der Portlets wird durch JavaServer Faces Technologie realisiert. Dank Einhaltung des JSR-168 Standards ist es möglich, einmal entwickelte Komponenten in eine andere Portalumgebung umzuziehen. Dies könnte eine Option sein, wenn die Entwickler von GridSphere das Projekt einstellen sollten oder wichtige technische Weiterentwicklungen nicht unterstützen.

Es ist zu beobachten, inwieweit die TRR Mitarbeiter tatsächlich von einer homogenen, personalisierbaren Benutzeroberfläche profitieren. Dies beinhaltet sowohl qualitative (Wird die Benutzeroberfläche als intuitiv bedienbar empfunden?) als auch quantitative Untersuchungen (Lässt sich eine messbare Steigerung der Effizienz im Arbeitsablauf feststellen?). Damit das volle Potenzial der Integrationsplattform ausgeschöpft werden kann, müssen die Nutzer in ihren neuen Möglichkeiten entsprechend geschult werden.

Ziel 3: Validierung des Konzepts durch Einbinden bestehender Funktionen in eine modulare Anwendungsoberfläche.

Die erfolgreiche Umwandlung des bestehenden pelican-Prototyps in eine Portalapplikation hat gezeigt, dass das entwickelte Konzept in der Praxis umsetzbar ist. Entscheidendes Erfolgskriterium ist der funktionierende Abruf der Plattform-Datenbank durch Ansteuerung eines caBIG Dienstes. Diese Funktion läuft jetzt für jede der drei Datenarten getrennt in Portlets ab. Bei der Programmierung der Portlets wurden explizite Änderungswünsche der TRR Mitarbeiter berücksichtigt: Eine neue Suchfunktion für Gensymbole und -lokation und eine variable Verknüpfung der verschiedenen Auswertungen wurden hinzugefügt.

Kritisch zu bewerten ist die nicht vollständige Unabhängigkeit der implementierten Module. Leider definiert der JSR-168 Standard keine Möglichkeiten für Inter-Portlet-Kommunikation (siehe Kapitel 2.4.3). Aus diesem Grund können die Portlets nur innerhalb einer Applikation untereinander Informationen austauschen. Das hat für die

pelican-Anwendung keine direkten Konsequenzen, aber aus dem Applikationskontext herausgelöst wäre die Verwendung einzelner Portlets nur eingeschränkt möglich. Beispielsweise reagiert das RNA-Modul spezifisch auf die Eingaben des Suche-Portlets und wäre, für sich genommen, externen Nutzern des Dienstes keine Hilfe. Mit der von GridSphere angekündigten Unterstützung des erweiterten Portletstandards JSR-286 muss die Thematik noch einmal untersucht werden. Möglicherweise lässt sich damit die Inter-Portlet-Kommunikation einfacher und generischer realisieren.

Eine große Hilfe für weitere Entwicklungen im pelican-Projekt ist die Identifikation der minimal notwendigen caBIG Bibliotheken. Durch die Reduzierung konnte die Anlaufzeit der Applikation deutlich verkürzt werden. Vorher häufig auftretende, mit den Bibliotheken in Verbindung stehende Konflikte wurden seitdem nicht mehr beobachtet.

5.3 GRENZEN DER ARBEIT

Die vorliegende Arbeit hatte nicht zum Ziel, eine einsatzreife Oberfläche für die pelican-Anwendung zu entwickeln. Es sollte das Konzept eines Portals erarbeitet und dieses praktisch mit einigen Beispielportlets validiert werden. Damit das System produktiv eingesetzt werden kann, müssen erst grundlegende Mechanismen zur sinnvollen Verarbeitung und Analyse der biomedizinischen Daten geschaffen werden. Bisher wurden nur die funktionalen Anforderungen weniger Projekte erfasst. Insbesondere projektübergreifende Auswertungsfragen sind derzeit noch in der Entwurfsphase. Mit der homogenen Darstellung verschiedener gekapselter Dienste wird lediglich die Präsentation der Programmlogik ermöglicht.

Es wird ebenfalls nicht tiefergehend untersucht, ob Portale die beste Möglichkeit zur Präsentation der angestrebten Plattform-Architektur darstellen. Obwohl in Kapitel 2.4 einige Argumente aufgeführt wurden, die dafür sprechen, gibt es durchaus Alternativen zur Portaltechnologie, zum Beispiel die von Google entwickelte OpenSocial Programmierschnittstelle.

Die Portalanwendung stellt Mechanismen zur Verfügung, um unterschiedliche Benutzerprofile und -gruppen einzurichten. Es wurde allerdings bisher kein Sicherheitskonzept umgesetzt, welches die Rechte der individuellen Nutzer(gruppen) festlegt oder Richtlinien für projektübergreifende Auswertungen in Bezug auf Datenschutz definiert.

Generell gibt es im Bereich der biomedizinischen Informatik noch viele Herausforderungen, die Berührungspunkte zur vorliegenden Arbeit aufweisen. Allein die unterschiedlichen Nomenklatursysteme für Gene oder deren mit jeder neuen Genomversion verschobene Position in der DNA, beeinflussen beispielsweise Funktionalität und Aussehen des Suche-Portlets. Die hier dokumentierte Herangehensweise bei der Entwicklung von Portlets soll jedoch als Hilfestellung dienen, beliebige Funktionen und neue Erkenntnisse in die pelican-Anwendung zu integrieren.

6 AUSBLICK

Im Dialog mit den Forschern des Verbundes eröffnen sich durch die entwickelte Portalanwendung ganz neue Möglichkeiten, um das Verständnis der beiderseitigen Arbeit zu fördern. Während der pelican-Prototyp eine ganz spezielle Fragestellung eines TRR-Projekts beantworten sollte, sind die Datenlieferungen der voneinander unabhängigen Portlets auch für andere Projekte interessant. Der modulare Aufbau veranschaulicht das Prinzip der Verknüpfung beliebiger Inhalte und Funktionen besser. Es können jetzt gemeinsam von Mitarbeitern des Z2-IT-Projekts und der medizinischen Projekte Ideen für weitere (projektübergreifende) Dienste entwickelt und umgesetzt werden. Die Standardkonformität des Portals erlaubt es, dabei auch auf extern programmierte Portlets mit relevanter Funktionalität zurückzugreifen.

Mit der Portalapplikation als Basis können sukzessive neue Komponenten in die Benutzeroberfläche integriert werden. Das Content Management System Alfresco wurde beispielsweise schon für den Einsatz in der Integrationsplattform konfiguriert. Es muss lediglich als Portlet eingebunden werden. Dabei kann der im Rahmen dieser Arbeit erstellte Leitfaden zur Portleterstellung in GridSphere Hilfestellung leisten. Die Integration eines CMS wäre ein großer Schritt in Richtung Wissensspeicher, in dem Forschungsergebnisse des TRR dokumentiert, veröffentlicht und diskutiert werden können. Daran lässt sich auch der für die Förderung durch die DFG notwendige vernetzte Charakter des Forschungsvorhabens gut demonstrieren.

Weitere Arbeit ist notwendig, um die Arbeitsabläufe der Forscher mit Hilfe der Anwendung besser unterstützen zu können. Die Verknüpfung der Dienste könnte beispielsweise über eine einfach zu bedienende Baukastenfunktion realisiert werden. Womöglich könnten daraufhin spezielle Dienste bei der Qualitätssicherung erstellter Dienstkompositionen unterstützen, indem sie beispielsweise die Einhaltung medizinischer Standards überprüfen. Bevor jedoch eine Automatisierung in irgendeiner Form stattfinden kann, muss innerhalb des Projektes ein umfangreiches Ontologiesystem etabliert werden, welches die vielen verschiedenen Begrifflichkeiten auf dem Gebiet der Krebsforschung formal ordnet. Die Suchfunktion ist beispielsweise wenig flexibel und würde von einer Erweiterung um natürlichsprachliche Abfragen profitieren. Die Technik dazu steckt noch in den Kinderschuhen, aber das von Berners-Lee et al. [2001] propagierte „Semantic

Web“ wird die Navigation im Netz grundlegend verändern.

In absehbarer Zukunft soll die pelican-Anwendung in das caGrid eingebunden werden. Damit ist es möglich, über die Applikation auf Dienste des Grid zuzugreifen. Umgekehrt bietet dieser Schritt auch Forschern weltweit die Möglichkeit, im TRR entwickelte Dienste zu nutzen. Durch so gewonnene Erkenntnisse leistet die Arbeit des SFB/TRR 77 einen wertvollen Beitrag zur Erforschung einer der tödlichsten Krankheiten unserer Zeit.

Abdelnur, A. and S. Hepper (2003). Java portlet specification version 1.0. Technical report, Sun Microsystems, Inc.

Akram, A., D. Chohan, X. D. Wang, X. Yang, and R. Allan (2005). A service oriented architecture for portals using portlets. Technical report, CCLRC e-Science Centre, CCLRC Daresbury Laboratory.

Balzert, H. (2001). *Lehrbuch der Software-Technik* (2. Aufl., 1. Nachdr. ed.). Lehrbücher der Informatik. Heidelberg: Spektrum Akad. Verl.

Berners-Lee, T., J. Hendler, and O. Lassila (2001). The semantic web. *Scientific American May 2001*, 34–43.

Booth, D., H. Haas, F. McCabe, E. Newcomer, M. Champion, and D. Orchard (2004). Web services architecture, w3c working group note 11.

Bosch, A. (2009). *Portlets und JavaServer Faces*. Frankfurt/Main: entwickler.press.

Browning, P. (2003). Portal Evaluation. Technical report, University of Bristol.

Dion Goh, B., A. Luyt, S.-Y. Chua, K.-N. Yee, H.-Y. Poh, and Ng (2008). Evaluating open source portals. *Journal of Librarianship and Information Science 40*(2), 81–92.

Dostal, W. (Ed.) (2005). *Service-orientierte Architekturen mit Web Services* (1. Aufl. ed.). Heidelberg; München: Elsevier, Spektrum Akad. Verl.

Erl, T. (2010). *SOA: Entwurfsprinzipien für serviceorientierte Architektur* (Repr. der Ausg. 2008 ed.). Programmer's Choice. München: Addison Wesley.

Foster, I. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications 15*(3), 200–222.

Gamma, E. (2003). *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software* (5., korrigierter Nachdr. ed.). Programmer's Choice. München: Addison-Wesley.

Goble, C. and R. Stevens (2008). State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics 41*(5), 687–693.

Großmann, M. and H. Koschek (2005). *Unternehmensportale: Grundlagen, Architekturen, Technologien*. Xpert.press. Berlin: Springer.

Gurzki, T. and H. Hinderer (2003). *WM 2003: Professionelles Wissensmanagement - Erfahrungen und Visionen: Beiträge der 2. Konferenz Professionelles Wissensmanagement - Erfahrungen und Visionen, 2. - 4. April in Luzern, Schweiz*, Volume 28. Bonn: Reimer, Ulrich and Abecker, Andreas.

Haux, R. (2004). *Strategic information management in hospitals: An introduction to hospital information systems*. Health informatics series. New York: Springer.

Hertl, M. (2005). Liver transplantation for malignancy. *The Oncologist 10*(4), 269–281.

Hinderer, H., T. Gurzki, A. Kirchhof, and J. Vlachakis (2004). „Was ist ein Portal?“ Definition und Einsatz von Unternehmensportalen. Technical report, Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO Stuttgart.

Knuth, D. E. (1974). Computer Programming as an Art. *Communications of the ACM 17*(12), 667–673.

Koru, A., K. Dongsong Zhang, E. Emam, and H. Liu (2009). An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering 35*(2), 293–304.

Koutsonikola, V. and A. Vakali (2004). Ldap: framework, practices, and trends. *IEEE Internet Computing 8*(5), 66–72.

Martone, M. E., A. Gupta, and M. H. Ellisman (2004). E-neuroscience: challenges and triumphs in integrating distributed data from molecules to brains. *Nature neuroscience 7*(5), 467–472.

McAllister, N. (2004). Opening up the code. *InfoWorld 26*(49), 42–48.

Melzer, I. (2008). *Service-orientierte Architekturen mit Web Services: Konzepte - Standard - Praxis* (3 ed.). München: Spektrum Akad. Verl.

Myers, D. G. (2008). *Psychologie* (2., erweiterte und aktualisierte Auflage. ed.). Springer-Lehrbuch. Heidelberg: Springer.

OASIS (2006). *Reference Model for Service Oriented Architecture Version 1.0*. OASIS.

Parkin, D. M., F. Bray, J. Ferlay, and P. Pisani (2005). Global cancer statistics, 2002. *CA: A Cancer Journal for Clinicians 55*(2), 74–108.

Saltz, J. (2006). cagrid: design and implementation of the core architecture of the cancer biomedical informatics grid. *Bioinformatics 22*(15), 1910–1916.

Stein, L. D. (2008). Towards a cyberinfrastructure for the biological sciences: progress, visions and challenges. *Nature Reviews Genetics 9*(9), 678–688.

Stufflebeam, D. L. (2001). Evaluation checklists: Practical tools for guiding and judging evaluations. *American Journal of Evaluation 22*(1), 71–79.

Thomas, M. P., J. Burruss, L. Cinquini, G. Fox, D. Gannon, L. Gilbert, G. v. Laszewski, K. Jackson, D. Middleton, R. Moore, M. Pierce, B. Plale, A. Rajasekar, R. Regno, E. Roberts, D. Schissel, A. Seth, and W. Schroeder (2005). Grid portal architectures for scientific applications. *Journal of Physics: Conference Series 16*, 596–600.

von Eschenbach, A. C. and K. Buetow (2006). Cancer informatics vision: cabig. *Cancer Informatics 2*, 22–24.

Zörner, S. (2006). *Portlets: Portalkomponenten in Java*. Frankfurt am Main: entwickler.press.

A

ANHANG

ANHANG A: NOTWENDIGE BIBLIOTHEKEN

Listing A.1: Für die Ausführung der pelican-Applikation mindestens notwendige caBIG Bibliotheken.

```
 Größe(B)       Dateiname

  533.613       acegi-security-1.0.4.jar
   43.579       asm-3.3.1.jar
1.599.570       axis-1.4.jar
  151.797       caGrid-core-1.3.jar
   50.370       caGrid-CQL-cql.1.0-1.3.jar
   31.380       caGrid-data-common-1.3.jar
  278.682       cglib-2.2.jar
  659.777       cog-jglobus-1.2.jar
  118.757       commons-beanutils-1.6.jar
   46.725       commons-codec-1.3.jar
  571.259       commons-collections-3.2.jar
  109.131       commons-digester-1.5.jar
   71.442       commons-discovery-0.2.jar
  261.809       commons-lang-2.4.jar
   52.915       commons-logging-1.1.jar
   89.967       jaxb-api-2.1.jar
  849.219       jaxb-impl-2.1.4.jar
3.127.225       jaxb-xjc-2.1.4.jar
   31.191       jaxrpc-1.1.jar
   12.143       sdk-grid-remoting-4.3.jar
    8.161       sdk-security-client-4.3.jar
  118.821       sdk-system-client-4.3.jar
2.593.850       spring-2.0.2.jar
  666.014       wsrf_core_enum-4.0.3.jar
```

ANHANG B: KRITERIENMATRIX

Produkt	GridSphere	
Version	3.2	
Quelle	http://www.gridsphere.org/gridsphere/gridsphere/guest/download/r/	
Datum	08.02.2011	
Hersteller	GridLab Projekt, 2002 EU-gefördert für 3 Jahre	
Lizenzkostenmodell	GridSphere Open Licencee, kostenlos	
Referenzinstallationen	MediGRID Projekt (portal.medigrid.de/gridsphere/gridsphere), HPC Europa, DGrid, PGrade, BIRN, Telescience, Australian Virtual Observatory, UK Escience	
		Score
Technische Rahmenbedingungen (Gewichtung: Faktor 3)		
Ist die Portalsoftware kompatibel mit Linux OpenSuse Version 11?	ja, da Java-basiert	1
Wird JSR 168 und JSR 286 unterstützt?	JSR 168	1
Wird das LDAP Protokoll unterstützt?	ja	1
Werden gängige DBMS unterstützt?	HSQL mitgeliefert, alle von Hibernate unterstützten DBMS möglich	1
Wird die Integration in eine Gridumgebung unterstützt?	ja, über GridPortlets	0.5
Zwischensumme (Aufsummierte Punktwerte x Gewichtung) max. 15		**13.5**
Support (Gewichtung: Faktor 2)		
Wie umfangreich ist die Dokumentation?	umfangreiche Guidesammlung (Installation, Portletentwicklung etc.)	1
Gibt es ein Support- oder Userforum?	Wiki und Mailingliste	0.5
Gibt es eine zusätzliche Programmierschnittstelle (API)?	ja	1
Zwischensumme (Aufsummierte Punktwerte x Gewichtung) max. 6		**5**
Layoutmanagement (Gewichtung: Faktor 1)		
Kann das Layout des Portals angepasst werden (Themes, Skins)?	ja, aber kein modernes Look&Feel. xml-basiert	0.5
Sind die GUI Komponenten verschiebbar und von variabler Größe?	ja, kein drag&drop	0.5
Können die Inhalte für jeden Benutzer angepasst werden?	ja	1
Kann das Layout für jeden Benutzer individuell angepasst werden?	ja	1
Zwischensumme (Aufsummierte Punktwerte x Gewichtung) max. 4		**3**
Sicherheit (Gewichtung: Faktor 3)		
Wird Single Sign-On unterstützt?	X.509 oder Shibboleth	1
Wie umfangreich ist das Passwortmanagement?	limitierte Loginversuche, Passwortrücksetzung, Captchas	1
Können Nutzer Gruppen zugewiesen werden?	ja	1
Können Nutzern und Gruppen spezielle Rechte zugewiesen werden?	ja	1
Werden Nutzeraktivitäten und Systemereignisse protokolliert?	Stack Traces können an Administrator gemailt werden	0.5
Zwischensumme (Aufsummierte Punktwerte x Gewichtung) max. 15		**13.5**
Besonderheiten (max. 5 Punkte)	keine	**0**
Gesamtwertung (max. 45 Punkte)		**35**

Abbildung A.1: Ausgefüllte Kriterienmatrix, Teil 1

LifeRay		Gateln (ehemals Jboss Portal / eXo Platform)	
6.0 GA3 (August 2010) Community Edition		3.1	
http://www.liferay.com/downloads/liferay-portal/available-releases		http://www.jboss.org/gatein/downloads	
09.02.2011		10.02.2011	
Liferay Inc.		Jboss by Red Hat, eXo Platform	
community open source, Lesser General Public License		Lesser General Public License (LGPL)	
caBIG, Cisco, Covisint, SunGard (Luminis 5) and CampusEAI		keine Information	
	Score		**Score**
ja	1	ja	1
ja, beide	1	ja, beide	1
ja, GUI-gestützt	1	ja, GUI-gestützt	1
alle gängigen	1	HSQL mitgeliefert, alle gängigen	1
nein	0	nein	0
	12		**12**
umfangreich, aktuell. Viele Guides (auch Flashvideos)	1	lediglich User und Reference Guide	0.5
Forum und Wiki	1	Forum und Wiki	1
ja, konnte aber wiederholt nicht über Webseite erreicht /geladen werden	0.5	Organization API, WSRP API	0.5
	5		**4**
ja, modernes Look&Feel	1	ja	
ja, drag&drop	1	ja, drag&drop	
ja	1	ja	
ja	1	ja	
	4		**4**
CAS, Siteminder, weitere unterstützt	1	CAS, JOSSO, OPEN SSO	1
Ablaufen von Passwörtern, Beschränkung von Loginversuchen etc.	1	in Dokumentation nur Passwortänderung beschrieben	0.5
ja	1	ja	1
ja	1	ja	1
nicht unterstützt	0	nein	0
	12		**10.5**
Integration von Alfresco (CMS) unterstützt, WSRP unterstützt	**2**	Token Service API, WSRP unterstützt	**2**
	35		**32.5**

Abbildung A.2: Ausgefüllte Kriterienmatrix, Teil 2

ANHANG C: DEPLOYMENT DESKRIPTOR DER PORTLETAPPLIKATION

Listing A.2: web.xml der Portletapplikation.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance"
        version="2.4"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee   http://java.sun.com/xml
            /ns/j2ee/web-app_2_4.xsd">

        <context-param>
                <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
                <param-value>server</param-value>
        </context-param>

        <context-param>
                <param-name>org.apache.myfaces.CHECK_EXTENSIONS_FILTER</param-name>
                <param-value>TRUE</param-value>
        </context-param>

        <context-param>
                <param-name>javax.portlet.faces.BridgeImplClass</param-name>
                <param-value>org.apache.myfaces.portlet.faces.bridge.BridgeImpl</
                    param-value>
        </context-param>

        <context-param>
                <param-name>javax.portlet.faces.renderPolicy</param-name>
                <param-value>ALWAYS_DELEGATE
                </param-value>
        </context-param>

        <listener>
                <listener-class>org.apache.myfaces.webapp.
                    StartupServletContextListener</listener-class>
        </listener>

        <filter>
                <filter-name>extensionsFilter</filter-name>
                <filter-class>org.apache.myfaces.webapp.filter.ExtensionsFilter</
                    filter-class>
                <init-param>
                        <description>Set the size limit for uploaded files.
                Format: 10 - 10 bytes
                        10k - 10 KB
                        10m - 10 MB
                        1g - 1 GB</description>
                        <param-name>uploadMaxFileSize</param-name>
                        <param-value>100m</param-value>
                </init-param>
```

```
                <init-param>
                        <description>Set the threshold size - files
                    below this limit are stored in memory, files above
                    this limit are stored on disk.

                Format: 10 - 10 bytes
                        10k - 10 KB
                        10m - 10 MB
                        1g - 1 GB</description>
                        <param-name>uploadThresholdSize</param-name>
                        <param-value>100k</param-value>
                </init-param>
        </filter>

        <filter-mapping>
                <filter-name>extensionsFilter</filter-name>
                <url-pattern>*.jsf</url-pattern>
        </filter-mapping>

        <filter-mapping>
                <filter-name>extensionsFilter</filter-name>
                <url-pattern>/faces/*</url-pattern>
        </filter-mapping>

        <servlet>
                <servlet-name>Faces Servlet</servlet-name>
                <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>

        <servlet>
                <servlet-name>PortletServlet</servlet-name>
                <servlet-class>org.gridsphere.provider.portlet.jsr.PortletServlet</
                    servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>PortletServlet</servlet-name>
                <url-pattern>/jsr/PELICAN</url-pattern>
        </servlet-mapping>

        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>*.faces</url-pattern>
        </servlet-mapping>

        <welcome-file-list>
                <welcome-file>search.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

Universität Heidelberg
Hochschule Heilbronn
Medizinische Informatik

Studiengang Medizinische Informatik
Masterstudiengang Informationsmanagement in der Medizin

Tino Noack (Name, Vorname)

163510 (Matrikelnummer)

Thema der Diplom-/Masterarbeit: Konzeption und prototypische Implementierung einer individualisierbaren Benutzeroberfläche für eine biomedizinische Webapplikation

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.
Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistung von folgenden Personen erhalten:

.....

.....

.....

.....

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und ist auch noch nicht veröffentlicht.

Heidelberg, 03. Mai 2011 (Ort, Datum)

..... (Unterschrift)